REPÚBLICA PORTUGUESA
FCT Fundação para a Ciência e a Tecnologia
NORTE2020 PROGRAMA OPERACIONAL REGIONAL DO NORTE
PORTUGAL 2020
UNIÃO EUROPEIA Fundo Social Europeu

Carina Sofia Marinho de Andrade A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing

UMinho | 2022

Universidade do Minho
Escola de Engenharia

Carina Sofia Marinho de Andrade

A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing

November 2022

Universidade do Minho
Escola de Engenharia

Carina Sofia Marinho de Andrade

A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing

Doctoral Thesis
Doctoral Program in Advanced Engineering Systems for Industry

Work done under the supervision of
Professor Maribel Yasmina Santos (PhD)
Professor Carlos Costa (PhD)

November 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho

Atribuição-NãoComercial

CC BY-NC

https://creativecommons.org/licenses/by-nc/4.0/

"A garden is a grand teacher. It teaches patience and careful watchfulness;

it teaches industry and thrift; above all, it teaches entire trust!''

(Gertrude Jekyll)

Acknowledgments

To my husband that always challenged me, supported me, loved me even when I was in a bad mood against everything and did not want to pursue my goals: thanks for pushing me to accomplish them. Thanks for all the care when I was down, thanks for all the motivational words that I ignored and that kept coming. Just thanks, I love you!

"I wanted to play tough
Thought I could do all just on my own
But even Superwoman
Sometimes needed Superman's soul"

('Helium' by Sia)

To my parents, sister and 'Tio de Vila Real', that always made all they could for me to have all the tools to be the best and to have better opportunities in my life. I will never forget that on my path, you were always there to support me!

" 'Ohana' means family. Family means nobody gets left behind or forgotten."

(Lilo & Stitch)

To my friends: Ana, thanks for understanding when I needed to talk and pick me for a walk in the park. Nuno, thanks for being always with me and always understand my pain even when I preferred that you did not. Margarida and Diogo, I am glad that I started this PhD because it has brought me both of you. JADE girls, I feel that I would not be here if you had not entered the Jade's adventure with me. Every weekend with you gave me the strength for one more week.

"Because of you, I laugh a little harder, cry a little less, and smile a lot more."

(Unknown)

To my supervisors: I just want to say that I tried to convince myself to give up many times, I just did not do it because of you. Such good and inspiring people deserved my effort! Thank you, Professor Maribel, to be a tech woman and a role model for me. Thank you, Professor Carlos, for showing me that we can be the best even when we are starting our career. If I completed this step in my life, it was for you.

"A professor takes a hand, opens a mind, and touches a heart"

(Unknown)

To all my friends and colleagues from LID4, I need to thank you for all the laughs, for all the endless lunches, for the sharing of knowledge, for the ideas, for your contributions to my growth in the area. A special thanks to the ones that made a direct contribution to the *IEB* project: this thesis would not be the same without your support and dedication. Thanks Carlos, José, Rebelo and Maria. I'm proud of your success in the field and I know that you will be the best.

"You don't have to be crazy to work here. We'll train you."

(Unknown)

To all the people from Bosch (specially Guilherme) that supported my work and help me each time that I have been lost in the giant company that is Bosch, thank you for understanding all the time that I asked for more and more information.

"All tasks are important, even the most modest."

(Robert Bosch)

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020, the Doctoral scholarship PD/BDE/135101/2017 and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 039479; Funding Reference: POCI-01-0247-FEDER-039479]. Moreover, some figures in this document use icons made by Freepik from www.flaticon.com.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

Nowadays, the whole industry makes efforts to find the most productive ways of working and it already understood that using the data that is being produced inside and outside the factories is a way to improve the business performance. A set of modern technologies combined with sensor-based communication create the possibility to act according to our needs, precisely at the moment when the data is being produced and processed. Considering the diversity of processes existing in a factory, all of them producing data, Complex Event Processing (CEP) with the capabilities to process that amount of data is needed in the daily work of a factory, to process different types of events and find patterns between them. Although the integration of the Big Data and Complex Event Processing topics is already present in the literature, open challenges in this area were identified, hence the reason for the contribution presented in this thesis. Thereby, this doctoral thesis proposes a system architecture that integrates the CEP concept with a rule-based approach in the Big Data context: the Intelligent Event Broker (*IEB*). This architecture proposes the use of adequate Big Data technologies in its several components. At the same time, some of the gaps identified in this area were fulfilled, complementing Event Processing with the possibility to use Machine Learning Models that can be integrated in the rules' verification, and also proposing an innovative monitoring system with an immersive visualization component to monitor the *IEB* and prevent its uncontrolled growth, since there are always several processes inside a factory that can be integrated in the system. The proposed architecture was validated with a demonstration case using, as an example, the Active Lot Release Bosch's system. This demonstration case revealed that it is feasible to implement the proposed architecture and proved the adequate functioning of the *IEB* system to process Bosch's business processes data and also to monitor its components and the events flowing through those components.

Keywords - Big Data, Complex Event Processing, Industry 4.0, Monitoring System

Resumo

Hoje em dia as indústrias esforçam-se para encontrar formas de serem mais produtivas. A utilização dos dados que são produzidos dentro e fora das fábricas já foi identificada como uma forma de melhorar o desempenho do negócio. Um conjunto de tecnologias atuais combinado com a comunicação baseada em sensores cria a possibilidade de se atuar precisamente no momento em que os dados estão a ser produzidos e processados, assegurando resposta às necessidades do negócio. Considerando a diversidade de processos que existem e produzem dados numa fábrica, as capacidades do Processamento de Eventos Complexos (CEP) revelam-se necessárias no quotidiano de uma fábrica, processando diferentes tipos de eventos e encontrando padrões entre os mesmos. Apesar da integração do conceito CEP na era de *Big Data* ser um tópico já presente na literatura, existem ainda desafios nesta área que foram identificados e que dão origem às contribuições presentes nesta tese. Assim, esta tese de doutoramento propõe uma arquitetura para um sistema que integre o conceito de CEP na era do *Big Data*, seguindo uma abordagem baseada em regras: o *Intelligent Event Broker* (*IEB*). Esta arquitetura propõe a utilização de tecnologias de *Big Data* que sejam adequadas aos seus diversos componentes. As lacunas identificadas na literatura foram consideradas, complementando o processamento de eventos com a possibilidade de utilizar modelos de *Machine Learning* com vista a serem integrados na verificação das regras, propondo também um sistema de monitorização inovador composto por um componente de visualização imersiva que permite monitorizar o *IEB* e prevenir o seu crescimento descontrolado, o que pode acontecer devido à integração do conjunto significativo de processos existentes numa fábrica. A arquitetura proposta foi validada através de um caso de demonstração que usou os dados do *Active Lot Release*, um sistema da Bosch. Os resultados revelaram a viabilidade da implementação da arquitetura e comprovaram o adequado funcionamento do sistema no que diz respeito ao processamento dos dados dos processos de negócio da Bosch e à monitorização dos componentes do *IEB* e eventos que fluem através desses.

Palavras-chave - *Big Data*, Indústria 4.0, Monitorização de Sistemas, Processamento de Eventos Complexos

Table of Contents

List of Figures

List of Abbreviations and Acronyms

In this document, abbreviations and acronyms are used, which are presented as follows:

3D - Three dimensions

ALR - Active Lot Release

API - Application Programming Interface

AR - Augmented Reality

BIDCEP - Big Data CEP

CEP - Complex Event Processing

CORE - Computing Research & Education

CPS - Cyber-Physical System

DR - Design Rule

DSMS - Data Stream Management Systems

ERA - Excellence in Research in Australia

FERARI - Flexible Event pRocessing for big dAta aRchItectures

GDM - Graph Data Model

HDFS - Hadoop Distributed File System

HTML - HyperText Markup Language

HTTPS - Hyper Text Transfer Protocol Secure SSL

IEB - Intelligent Event Broker

IoT - Internet of Things

KPI - Key Performance Indicator

ML - Machine Learning

MLML - Machine Learning Models Lake

REST - REpresentational State Transfer

SLAM - Simultaneous Localization And Mapping

SQL - Structured Query Language

Chapter 1. Introduction

This chapter introduces the context, motivation and contributions of this doctoral thesis as well as the structure of this document, which is based in several chapters that present the contributions of research papers that address the research objectives defined for this doctoral thesis.

1.1 Industry 4.0 at Bosch Car Multimedia

To introduce the Industry 4.0 term, an historical background needs to be given regarding the evolution of the industry over time. In this sense, Figure 1-1 summarizes the main points associated to each industrial revolution since the 18th century.

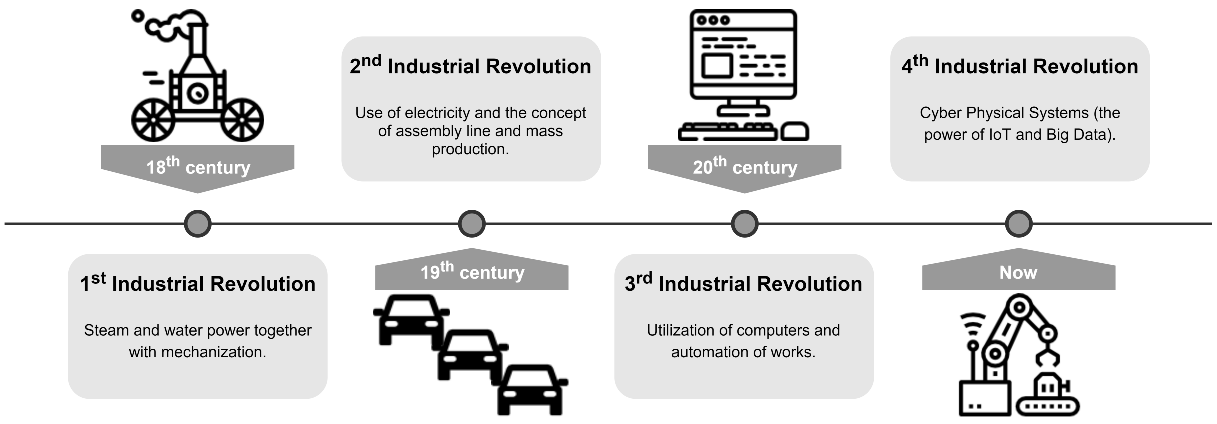

18th century
1st Industrial Revolution
Steam and water power together with mechanization.
2nd Industrial Revolution
Use of electricity and the concept of assembly line and mass production.
19th century
20th century
3rd Industrial Revolution
Utilization of computers and automation of works.
4th Industrial Revolution
Cyber Physical Systems (the power of IoT and Big Data).
Now

Figure 1-1. The four industrial revolutions

During the 18th century, the first industrial revolution started with the usage of the water and steam power that allowed for the mechanization of the work. Already in the 19th century, the second revolution brought the concepts of assembly lines and mass production using electricity. Regarding the third industrial revolution, it was focused on integrating information technology into production processes.

Currently, we are living the fourth industrial revolution, and we can debate about what are, in fact, the main topics of this concept. It is perceived that the concept was mentioned for the first time in 2011 at the Hannover Fair in Germany (Pfeiffer, 2017). This concept covers the main technological innovations applied to production processes in the field of automation, control and information technologies (Hermann, Pentek, & Otto, 2016). The foundation of the Industry 4.0 indicates that the organisations can control the production processes autonomously along with the value chain, when the machines are connected as well as their systems and assets. Considering the connection between systems, the Industry 4.0 framework argues that the organizations will have the capacity and autonomy to schedule maintenance, predict failures and adapt themselves to new requirements and unplanned changes in the production processes (Jazdi, 2014).

Bosch Car Multimedia is a division of Bosch dedicated to automotive electronics focusing on the development and production of sensors and multimedia products for cars. Being this doctoral thesis carried out in the context of Bosch Car Multimedia, the concept of Industry 4.0 is inherent to the topics being explored at Bosch. As a brief context, Bosch Car Multimedia has several buildings that make up the Bosch factory, with several machines producing the products and a vast amount of data being generated every moment.

Taking into account this overview over the industrial context in which this thesis is being developed, the main goal of this thesis is related to the use of relevant data available in Bosch at the time that it is being produced in the factory, to provide insights in real time about the actions that must be made to avoid serious consequences in the products being produced or to plan the machines' maintenance, for example, among other relevant use cases. For that reason, in this doctoral thesis, it is proposed to advance state of the art in this field integrating a Complex Event Processing (CEP) System in Big Data contexts.

1.2 Main Contributions of this Work

The CEP concept exists since the 90's but, with the current amount and variety of data that is being produced by several machines and different technologies, it needs to be scaled up to keep up with the needs in terms of storage, processing, time frames, prediction and prescription, connections to recent technologies, among others.

Despite the fact that some works mentioned throughout this doctoral thesis (Ioannis Flouris et al., 2016; Hadar, 2016) try to connect the CEP concept with the vast amount of data available nowadays, to the best of our knowledge, those works do not significantly detail how the components of a possible solution are related and communicate between them; they do not present ways to integrate Machine Learning (ML) algorithms in the rules verification; and they also do not identify current ways to monitor a CEP in Big Data considering its constant growth. In this context, this work aims to provide a new vision over the CEP concept, dealing with all the events being produced by factory machines in a production plant and providing the possibility to take actions, immediately, over the machines and products being produced on those machines and, at the end, increase the productivity of the organization. This system has the name of Intelligent Event Broker (*IEB*) and was detailed in several chapters, from the identification of the needs found in this area to the significantly detailed specification of the proposed system architecture and the explanation of the transversal demonstration case explored through different perspectives in those chapters. This demonstration case is based on Bosch's Active Lot Release (ALR) system that supports the quality control processes during the production and packaging of the products. Although the demonstration case is the same across all the chapters, in each one of them, it focuses on the specific aspects that need to be demonstrated and validated according to the *IEB* components being proposed and developed in that specific work.

The main architecture of the *IEB* includes the detail on how its components should communicate with each other and how the events must be integrated and flow through the system. Apart from that, it also presents the inclusion of a ML component to allow the execution of algorithms over the events arriving at the system, when the event processing rules are being verified, in order to predict and recommend actions regarding the machines or products being produced. Although this component is considered in the *IEB* architecture and briefly explained how it should be implemented, the same was not integrated in the demonstration case due to time constraints. However, it is our conviction that its inclusion in the proposed architecture already provides the specification that researchers and practitioners need on how to include and explore this component in a context combining CEP and Big Data concepts. The contribution related to the monitoring system for the *IEB* was fully explored with three research papers that detailed the design proposed and how it should be implemented to ensure the controlled growth of the *IEB* using a demonstration case for that.

1.3 Document Structure

Taking into consideration that the chapters of this thesis address the challenges and contributions of different research papers, each chapter describes a research paper, starting with a brief introduction to the chapter so that the same can be properly understood within the context of this document. Each chapter describes a paper just with the necessary adjustments to ensure the document consistency in this doctoral thesis document. Those adjustments are limited to the form and not to the semantics and contributions. Figure 1-2 gives an overview on the relationship of each main objective (detailed in the following chapter) of this doctoral thesis and the chapters included in this document, which content describes the published research papers.

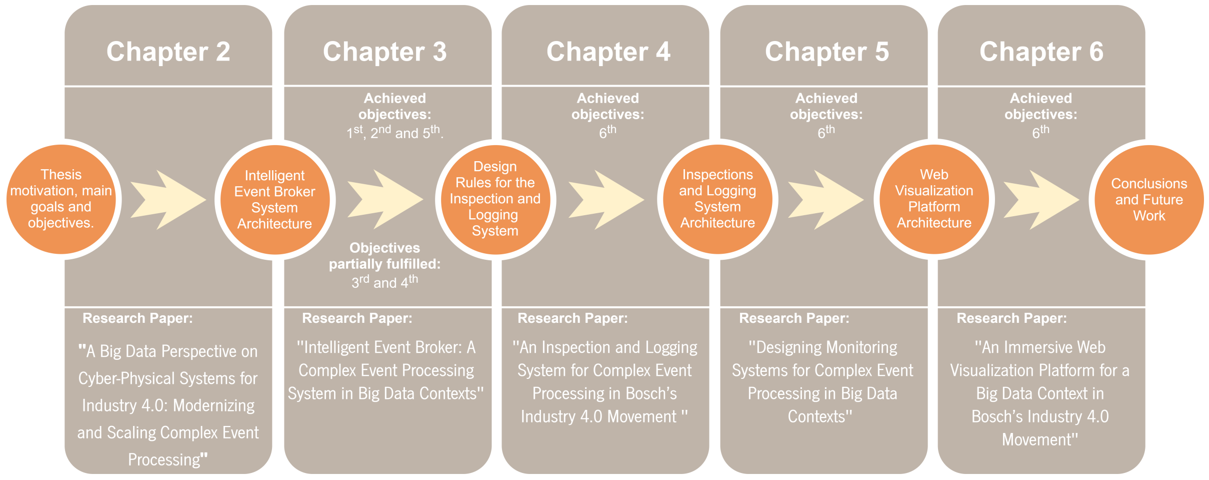

Chapter 2
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Achieved objectives: 1st, 2nd and 5th.
Achieved objectives: 6th
Achieved objectives: 6th
Achieved objectives: 6th
Thesis motivation, main goals and objectives.
Intelligent Event Broker System Architecture
Design Rules for the Inspection and Logging System
Inspections and Logging System Architecture
Web Visualization Platform Architecture
Conclusions and Future Work
Objectives partially fulfilled: 3rd and 4th
Research Paper: "A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing"
Research Paper: "Intelligent Event Broker: A Complex Event Processing System in Big Data Contexts"
Research Paper: "An Inspection and Logging System for Complex Event Processing in Bosch's Industry 4.0 Movement "
Research Paper: "Designing Monitoring Systems for Complex Event Processing in Big Data Contexts"
Research Paper: "An Immersive Web Visualization Platform for a Big Data Context in Bosch's Industry 4.0 Movement"

Figure 1-2. Overview on the thesis contributions

This document is structured as follows:

- Chapter 2 consists of an introductory chapter that presents the motivation for this research work in the Bosch Car Multimedia context and in the scientific context on integration of CEP and Big Data. Moreover, the chapter presents the main goal, objectives and expected contributions for this work, as well as the research methodology that was used during the research process of this thesis.

- Chapter 3 presents the overall architecture of the *IEB*, as well as the demonstration case based on data from Bosch's ALR system.

- Chapter 4 discusses the design rules proposed for the definition of the data model for the Inspection and Logging component of the *Mapping and Drill-down System* of the *IEB* (briefly presented in Chapter 3). This chapter also complements the demonstration case by continuing to leverage the ALR data to validate the feasibility of the implementation of the Inspection and Logging component.

- Chapter 5 details the implementation of the logging/monitoring system that will work in parallel with the *IEB* and explains how the relevant metadata of the *IEB* is being collected, processed, and stored for further analysis in the Web Visualization Platform. In this chapter, the demonstration case continuous to be extended by focusing on assessing the suitability of the *IEB* logging/monitoring system.

- Chapter 6 is focused on the architecture of the Web Visualization Platform that is a part of the *Mapping and Drill-down System* and that will be used to analyse the metadata being collected on the *IEB*. In this final work, the demonstration case presented here takes a different perspective, by focusing on the end-user visualization leveraged by the *IEB* Web Visualization Platform.

- Chapter 7 summarizes the main contributions of this work and highlights relevant future work.

Chapter 2. Methodological Context

This chapter[1] introduces the scope of this doctoral thesis, presenting a brief state- of- the- art of the topics of interest, the main goal, objectives and expected contributions of this thesis, as well as the research methodology and a brief overview of the proposed approach.

2.1 Introduction

Currently, several business sectors are trying to catch up with the Big Data era, namely dealing with multiple data sources, in different formats, and different velocities. With the Internet of Things (IoT) proliferation, organizations can have their business processes complemented with sensors that produce event data contributing to monitoring their processes. Considering the existence of these hardware capabilities, some technologies to handle all the data that is constantly being produced are required, such as Spark[2], Druid[3] or Storm[4], which are of major relevance for processing, aggregating or analyzing streaming data in real time. Besides the technologies needed to handle the streaming data, some technologies for Big Data Warehousing (e.g. Hive[5]) can be relevant to complement the streaming data analysis. Considering these technologies, the system's scalability regarding data processing and storage is guaranteed. However, concepts such as Complex Event Processing (CEP) and rule-based technologies (e.g., Drools[6]) are called into this context to allow the processing of different types of events, finding patterns between them and using rules for that, reflecting the business requirements identified in the context of each organization. The integration of these concepts and technologies is already mentioned in

---

[1] This chapter integrates the content of the research paper "A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing", published at Doctoral Consortium of the 31st International Conference on Advanced Information Systems Engineering (CAiSE'2019) (Andrade, 2019). **Acknowledgements.** This work has been supported by FCT – Fundação para a Ciência e Tecnologia, Projects Scope UID/CEC/00319/2019 and PDE/00040/2013, and the Doctoral scholarship PD/BDE/135101/2017. This paper uses icons made by Freepik, from www.flaticon.com

[2] https://spark.apache.org/

[3] http://druid.io/

[4] http://storm.apache.org/

[5] https://hive.apache.org/

[6] https://www.drools.org/

some identified works, but it is not considered that they can be complemented with Machine Learning (ML) techniques that will allow the system to make predictions or recommendations using pre-determined ML models over events that arrive at the system. In addition, no other work mentions the importance of a complete and innovative visualization component for monitoring this type of systems, which is being considered in this work.

In this context, the goal of this doctoral thesis is the proposal of a logical and technological system architecture that provides to organizations the capability of using all their event data in real-time fashion, considering: i) business requirements that should be easily integrated into the system; ii) powerful ways of doing predictions and recommendations to improve daily operations; iii) system self-management and monitoring to prevent uncontrolled growth of the system. As can be observed in Section 2.2, the idealized system architecture addressing the mentioned points was not identified in the literature review, reason why this work discusses the proposal of a CEP system for the Big Data era.

This chapter is divided as follows: Section 2.2 presents the state-of-the-art; Section 2.3 mentions the expected contributions; Section 2.4 dissects the research methodology; Section 2.5 highlights the proposed approach and current results; and, Section 2.6 summarizes the presented and future work.

2.2 State-of-the-Art

A CEP system can be described as a system that analyses events through different perspectives like pattern matching or inference. In these processes, the system can filter and aggregate the relevant information, complementing it with external data (Leavitt, 2009). Besides the concept itself, Chakravarthy & Qingchun (2009) presents CEP systems as being a challenge for the Data Stream Management Systems, considering that besides processing the data in real-time, it is necessary to take action over it. The Rapide project (Luckham, 1996; Luckham & Vera, 1995) is often considered as the first work to explore the CEP concept (Cugola & Margara, 2012; Leavitt, 2009). This project started in the 90s and provided the capability of identifying the temporal and causal relationship among events. However, the current amount of available data requires improvements to this concept, adapting it to Big Data environments.

In this context, and analyzing the existing architectures that aim to integrate the CEP and Big Data concepts, the work of Hadar (2016) uses the recognized Lambda and Kappa architectures as the base for the proposal of BiDCEP, an architecture that integrates CEP and Big Data Streaming concepts. A relevant point mentioned by the authors is the relevance of the IoT concept, which 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 highlighted in the FERARI project context (Ioannis Flouris et al., 2016), with a prototype for real-time CEP that process vast amounts of event data in a distributed way. These two architectures share some principles, such as the components responsible for the connection to the data sources; the components associated with the event processing; and, the components related to the data consumers.

The main properties that should be considered in the development of a Big Data CEP system are mentioned by I. Flouris et al. (2017): parallelism, elasticity, multi-query and distributed resources. However, the authors also mention that, although there are some works that try to address these issues, the integration of CEP and Big Data technologies is still something significantly unexplored.

Nevertheless, more than integrating CEP in Big Data contexts, some works (Babiceanu & Seker, 2015; Dundar, Astekin, & Aktas, 2016; Krumeich, Jacobi, Werth, & Loos, 2014) emphasize the relevance of combining Big Data, CEP, and IoT to support the manufacturing industry through Cyber-Physical Systems (CPSs). The work of Babiceanu & Seker (2015) proposes a framework for a Manufacturing CPS that considers the physical world (e.g., manufacturing facilities and shop-floor resources); the cyber world (e.g., simulation and prediction models); and, the interface between these two worlds (e.g., sensor networks and structured and unstructured data). In this case, the CEP system is a component in the cyber world, responsible for processing events and return results in (near)real-time that could provide operational visibility and awareness for the manufacturing system. In Krumeich et al. (2014), the authors explore the use of event-based predictions for manufacturing planning and control. In this case, sensors installed in the manufacturing plant are the data source for the CEP system that, combining the events with historical data, will provide the possibility of achieving the event-based prediction models for production planning and control. The work of Dundar et al. (2016) proposes a framework that can be applied to monitor the status of a CPS through the use of IoT data. This framework presents a Publish-Subscribe Messaging System that receives the data for further identification of meaningful events in a rule-based CEP System (running in a distributed way). The processing results are published in the self-healing mechanism and predictive maintenance component for the execution of the actions previously defined.

2.3 Expected Contributions

Although the main concepts and components of the analyzed architectures are being considered in the architecture to be proposed in this doctoral thesis, current contributions: i) do not clearly and rigorously refer or detail how CEP and Big Data concepts and technologies can act together for distributed data, rules and events processing with (near)real-time aggregations and Key Performance Indicators (KPIs) at data ingestion time; ii) do not combine batch data arriving from a Big Data Warehouse (Costa & Santos, 2018), complementing the event data that arrives in a streaming way and bringing more value to the results and actions of the system; iii) do not discuss techniques similar to the ML models lake component (presented in section 2.5), which can be significantly helpful for patterns discovery and it is identified as a major gap in these systems (Tawsif, Hossen, Emerson Raja, Jesmeen, & Arif, 2018); and, iv) do not consider the relevance of monitoring the functioning and evolution of this type of systems that can quickly become untraceable in Big Data contexts.

Beyond the current results presented in Section 2.5, this work reveals to be of significant relevance to several contexts considering that the proposed system architecture should be generic to be applied in different areas, such as industry, smart cities, agriculture, among others, where several data sources are involved (as it is intended to be shown during this thesis, using different demonstration cases). Consequently, this system can be considerably helpful for the monitoring of business processes and events, also using complementary data, to prevent possible problems through the capabilities of its *Predictors and Recommenders* component, which is directly linked to the actions that could be triggered, components not seen in other identified works. In industrial contexts, the role of this system can be easily identified in the manufacturing shop floor where machines and other interconnected devices are increasing, or even for the analysis of customers' reviews in social media, for example. In this context, this system can, for example, monitor the production data and, if some business rule is activated (e.g., for a defective product), predict if the next products will also be defective products and, if true, stop the production machine. Also, the system can predict and prevent a brand crisis related to an organization's publication or a defective product, which is generating negative comments in social media. In smart cities and agricultural contexts, the goal is fundamentally the same, i.e., use the data that is being produced by several sources and process it in (near)real-time, sending, for example, an accident warning to a street screen or changing the amount of water for irrigation due to a temperature change.

2.4 Research Methodology

This work follows a design science (Hevner, March, Park, & Ram, 2004) research approach with the goal of extension of the boundaries of human or organizational capabilities through the creation of new and innovative artifacts, in this case, proposing a solution for organizations pursuing (near)real-time decisions and automated actions based on Big Data streams and CPSs. In this context, Hevner et al. (2004) presents a set of guidelines to conduct design science research in Information Systems: i) propose an artifact to address an organizational problem; ii) understand the relevance of the problem that is being solved; iii) evaluate the utility, quality and efficacy of the proposed artifact; iv) provide clear and verifiable research contributions; v) apply rigorous methods on the artifact development and evaluation; vi) consider the design of the artifact as a search process utilizing the available means to fulfil the goals and satisfy the constraints; and, vii) present the research to technology practitioners and researchers, as well as management-oriented audiences. Moreover, Hevner et al. (2004) considers that the Information Technology (IT) artifacts, resulting from a design science research process, can be: constructs (vocabulary and symbols); models (abstractions and representations); methods (algorithms and practices); and, instantiations (implementations and prototype systems).

Therefore, the widely recognized Design Science Research Methodology for Information Systems (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007) is used in this doctoral thesis, considering an objective-centred approach for the designing of a logical and technological system architecture that must meet the following objectives:

1) Handle Big Data produced by several sources inside and outside the organizations (e.g., data from production lines, cars, citizens, smartphones, among others);

   a) Consider several possible data sources and their interfaces' differences, in order to design a system that ensures that new data sources can be easily added;

   b) Consider the volume, variety and velocity of the data that could arrive at the system, in order to define its scalability, multi-query, parallelism and distributed resources characteristics.

2) Consider the business requirements and indicators defined by organizations, besides the data itself;

3) Process the data within the time frame needed for several decision makers, with organizations providing inputs regarding the latency for the different system use cases, as this can be used to evaluate the timeliness of the automated actions triggered by the system;

4) Provide predictions and recommendations for the organization's daily activities;

   a) Design and evaluate an adequate system architecture to efficiently integrate predictive and prescriptive ML models into the streams of data/event processed by the system, providing high throughput and low latency predictions/prescriptions.

5) Autonomously execute adequate actions to avoid considerable problems for the organization (e.g., stop a production machine);

   a) Design and evaluate the most adequate way to communicate with external systems, executing automated actions directly related to the business requirements (rules) and indicators defined by the organization.

6) Consider the relevance of self-management and monitoring, preventing the uncontrolled growth of the system with the constant monitoring and visualization of what happened in the system (e.g., Which producers are introducing more data into the system? What are the most triggered actions?).

a) Design and evaluate the monitoring system and the visualization platform that should consider strategic endpoints in which data is collected and provide user-friendly analysis of the system status (e.g., immersive and drill-down visualization, virtual or augmented reality).

Considering these objectives, some metrics were identified as relevant for the system evaluation: scalability and complexity when integrating new data sources, producers and consumers in the system; number of events produced/consumed per second; number of rules, actions and ML models verified, triggered and applied per second, respectively; average response time of the application of ML models; and, usefulness of the analyses made available in the system's monitoring platform. These metrics can be evaluated in the future considering the organizations' requirements and/or using baselines and guidelines identified in the literature review (e.g., throughput, latency and scalability benchmarks).

The results of a first iteration on the Design and Development phase of the research methodology are presented in Section 2.5, and the 1st, 2nd, 3rd 4th, and 5th objectives discussed above were already completely or partially fulfilled through a real-world prototype implementation at Bosch Car Multimedia Portugal, although it is further detailed in Chapter 3.

2.5 Proposed Approach and Current Results

This section presents the first iteration of the Design and Development phase of the research methodology explained in Section 2.4, using a proof of concept based on the Active Lot Release application from Bosch Car Multimedia Portugal as a demonstration case. Taking into consideration the organization's needs and the gap found in the literature review, a system architecture is being proposed to fill these needs. This system, named Intelligent Event Broker, aims to represent a Big Data-oriented CEP system that combines a collection of software components and data engineering decisions, integrated to ensure their usefulness, efficiency and harmonious functioning, in order to process the events that arrive in the system.

The proposed architecture for the Intelligent Event Broker (Figure 2-1) considers a vast number of components for dealing with the volume, variety and velocity of the data:

1. *Source Systems*: the system architecture should be prepared to receive data from several sources: relational, NoSQL or NewSQL databases, IoT Gateways or (Web)Servers, and even components of the Hadoop ecosystem, such as Hive Tables or HDFS files;

2. *Producers*: to ensure that all the *Source Systems* identified in 1. can be integrated into the system, regardless of their communication interfaces, the *Producers* component is proposed to standardize the collection of events entering the system. Kafka[7] (a distributed streaming platform) is proposed for the deployment of this component;

3. *Broker Beans*: the events collected in the *Source Systems* by the Kafka *Producers* are serialized into the form of classes that define the several business entities existing in the system – *Broker Beans*;

4. *Brokers*: events serialized into *Broker Beans,* can be published by the *Producers* into a Kafka topic that is stored in a cluster of Kafka *Brokers*;

5. *Event Processor*: events are subscribed by the *Event Processor* (Kafka Consumers that are embedded into Spark Applications) that are always waiting for processing the events arriving at the system, regardless of their frequency;

6. *Complementary Data*: in addition to the events published in the topics, the *Event Processor* can use *Complementary Data* from the *Source Systems,* if useful for the event processing, providing additional and relevant information;

7. *Rules Engine*: includes the defined rules that represent the Business Requirements (with Strategical, Tactical or Operational rules). This work is usually associated to a Data Engineer that creates the rules that represent the business needs. Here Drools is used to store the rules that will be then translated by the *Event Processor*, using a custom-made integration of Spark and Drools, based on previously explored paths by the technical community (Ganta, 2015).

[7] https://kafka.apache.org/

8. *Triggers*: connectors to the *Destination Systems*, execute the actions previously defined for the rules when the condition is evaluated as being true.

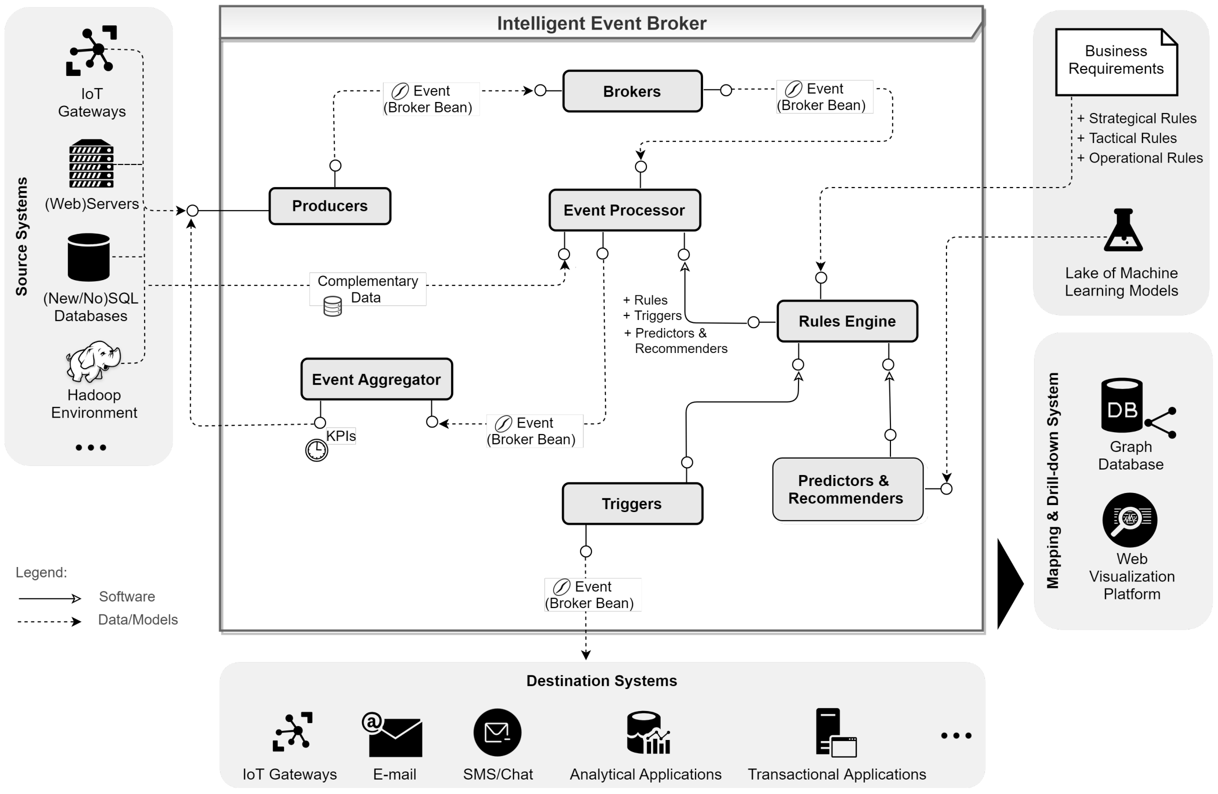


Figure 2-1. First version of the proposed Intelligent Event Broker system architecture

9. *Destination Systems*: the results of processing the events can be sent to, for example, *IoT Gateways* that can activate an actuator, *Text* or *E-mails Messages,* or even *Transactional* or *Analytical Applications*;

10. *Predictors and Recommenders*: the concept of a *Lake of ML Models*, which are trained beforehand, is proposed in this system architecture as being of major relevance. Allow the application of those ML models to the data that is being processed, providing the capability to predict occurrences or recommend actions based on the events that are arriving at the system;

11. *Event Aggregator*: stores the raw event data (events that arrived at the system) or processed event data (the events processing result, such as results from the *Predictors and Recommenders* component) used to calculate the KPIs relevant to the business. This component is supported by Druid, a columnar storage system useful for aggregating event data at ingestion time (Correia, Santos, Costa, & Andrade, 2018; Yang et al., 2014);

12. *Mapping and Drill-down System*: allows the constant monitoring of the Intelligent Event Broker and includes:

    12.1. *Graph Database*: stores the relevant metadata of the Intelligent Event Broker, allowing the exploration of the flows of the events in the Intelligent Event Broker;

    12.2. *Web Visualization Platform*: provides an interactive and immersive visualization regarding the Intelligent Event Broker metadata, stored in the *Graph Database,* taking into account the various implementation contexts of the system.

The demonstration case already implemented in Bosch Car Multimedia Portugal plant context (Chapter 3 and Andrade, Correia, Costa, & Santos (2019)) uses data from ALR System that supports the quality control used in the manufacturing and packaging processes. This system is based on rules that are applied to the products contained into lots before they are shipped to customers. The ALR system provides a stream of events that contain information about the quality control process, being considered, at this point, the lot identification, its packaging date, the production line, and the status ("*Valid*" or "*Invalid*" lot).

Therefore, for this demonstration case, one *Operational Rule* and two *Tactical Rules* were defined, as well as their own *Triggers* (store data into Cassandra for further analysis and send an *E-mail Message* to a stakeholder) that are activated if the result for the rule condition is true. Considering these two types of rules, two types of dashboards were created on the *Analytical Application*, one oriented for operational analysis and the other one oriented for a more tactical point of view. These dashboards, and more details about the proposed aproach and current results, can be seen in Chapter 3 and Andrade et al. (2019).

2.6 Chapter Summary

Considering the current evolution of the industrial world, this document presents the status and main research goals of a doctoral thesis that is dedicated to exploring the Big Data and CEP concepts in the Industry 4.0 movement. The context for the emergence of this topic of interest, as well as the research agenda, were explained in this work, and the state-of-the-art was detailed to highlight the contributions that distinguish this work from the already existing contributions. From the methodological point of view, the research methodology was presented, and the current status of the proposed contribution was also highlighted.

This chapter also presents a summary of the first version of the system proposed and implemented with the Bosch Car Multimedia Portugal demonstration case, that is fully detailed in Chapter 3. With this briefly explanation we can conclude that the 1st, 2nd and 5th objectives discussed in Section 2.4 were fulfilled and the 3rd, 4th objective were partially fulfilled: the 3rd one was not completely tested in terms of timeframes expected by the organization and the 4th was only considered in the proposed architecture but not implemented in a demonstration case.

However, with this prototype, a CEP system in Big Data contexts that reveals its adequacy to the problem and contains several components already developed (e.g., Data Producers and Consumers, Rules and Triggers with the business requirements, an Event Aggregator and an Analytical Application as Destination System) is presented to practitioners and researchers.

Chapter 3. Intelligent Event Broker Architecture

This chapter[8] presents the architecture of the Intelligent Event Broker, being the same divided in the following sections: the introduction of the related work; the proposal of the system architecture and its software packages and classes; and a demonstration case based on data from Bosch's Active Lot Release system in the Braga factory. This work served the purpose of accomplishing (completely or partially) the 1st, 2nd, 3rd, 4th and 5th objectives mentioned in Chapter 2.

3.1 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, Wahlster, & Helbig, 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)

[8] This chapter integrates the content of the research paper "Intelligent Event Broker: A Complex Event Processing System in Big Data Contexts", published at The Americas Conference on Information Systems (AMCIS'2019) (Andrade, Correia, Costa, & Santos, 2019). The contributions of Carina Andrade for this work were focused on the design of the system architecture, as well as the description of the communication and flow of the data between the system architecture components, actively participating in the implementation of the demonstration case. **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

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 and considering the organization's business requirements; 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, iii and iv were fulfilled, although the timeframes were not fully discussed with the organization and the predictions and recommendations were not implemented in the demonstration case, but considered in the proposed architecture. In summary, the solution reveals an adequate effectiveness, considering the success of the presented proof-of-concept. The efficiency and scalability of the solution are already a concern of the current architecture when choosing adequate and scalable big data technologies to support the components of the architecture.

This chapter is structured as follows: the second section (3.2) presents the related work; the third section (3.3) introduces the proposed system architecture and its software packages and classes; the fourth section (3.4) presents a demonstration case based on data from Bosch's ALR application in the Braga plant shop floor in Portugal; the fifth section (3.5) summarizes the presented work.

3.2 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 & 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 & Margara, 2012).

The work of Chakravarthy & 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 & Vera, 1995) is often unanimously considered as being the first one that explores this concept (Cugola & 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 & 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, Artikis, Deligiannakis, & Garofalakis (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 Ioannis 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 I. 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 & 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, Shi, & Khan, 2018) also aims to connect the CEP concept with Big Data, as well as the work of Saenko, Kotenko, & Kushnerevich (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 chapter. 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 & 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.

3.3 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 3-1) and the structure of software packages and classes (Figure 3-2) of the Intelligent Event Broker.

3.3.1 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 ("Apache 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*.

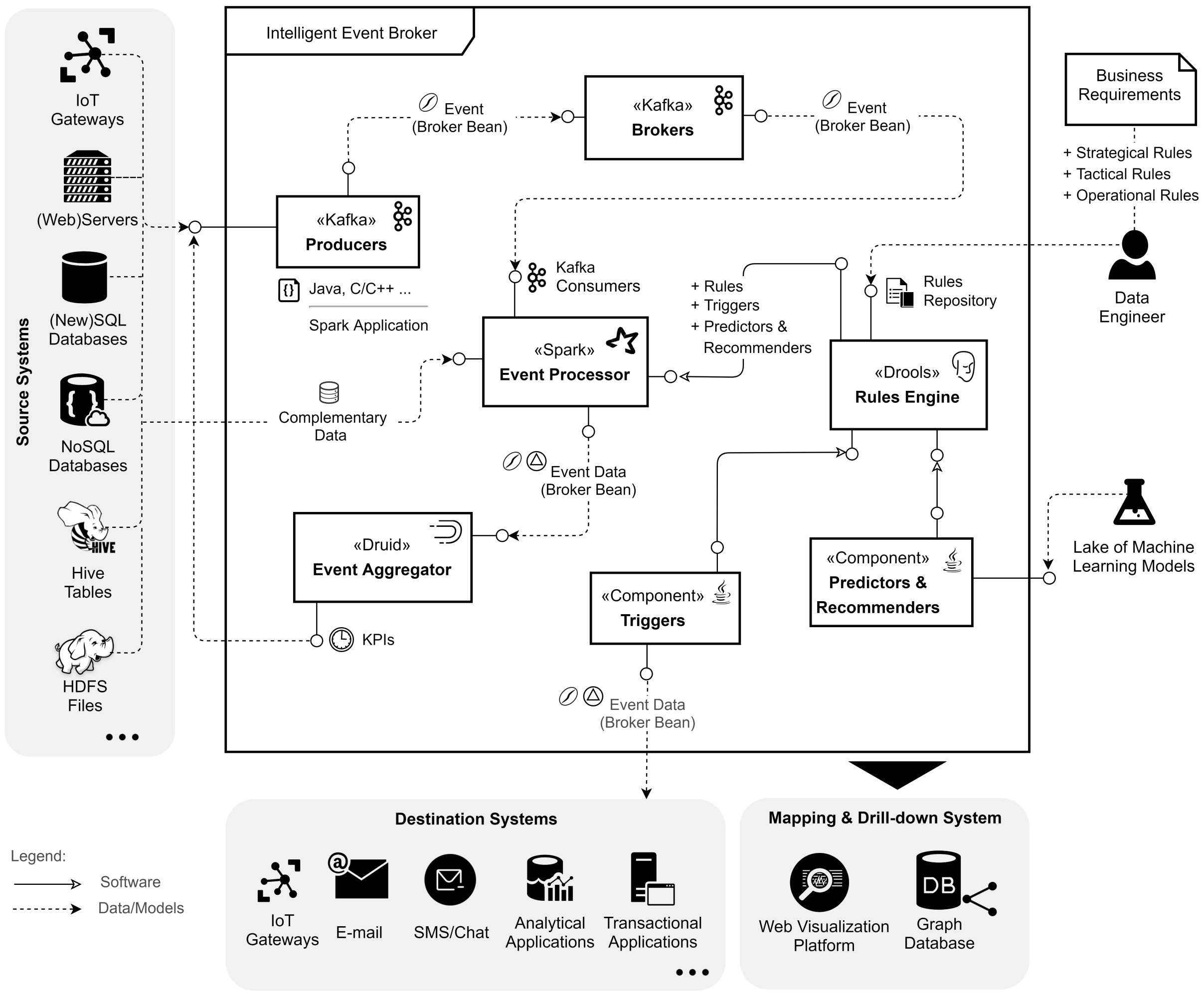


Figure 3-1. Intelligent Event Broker system architecture

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 - Business Rules Management System," 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*.

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.

3.3.2 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 3-2, as well as the components in Figure 3-1, 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 3-1).

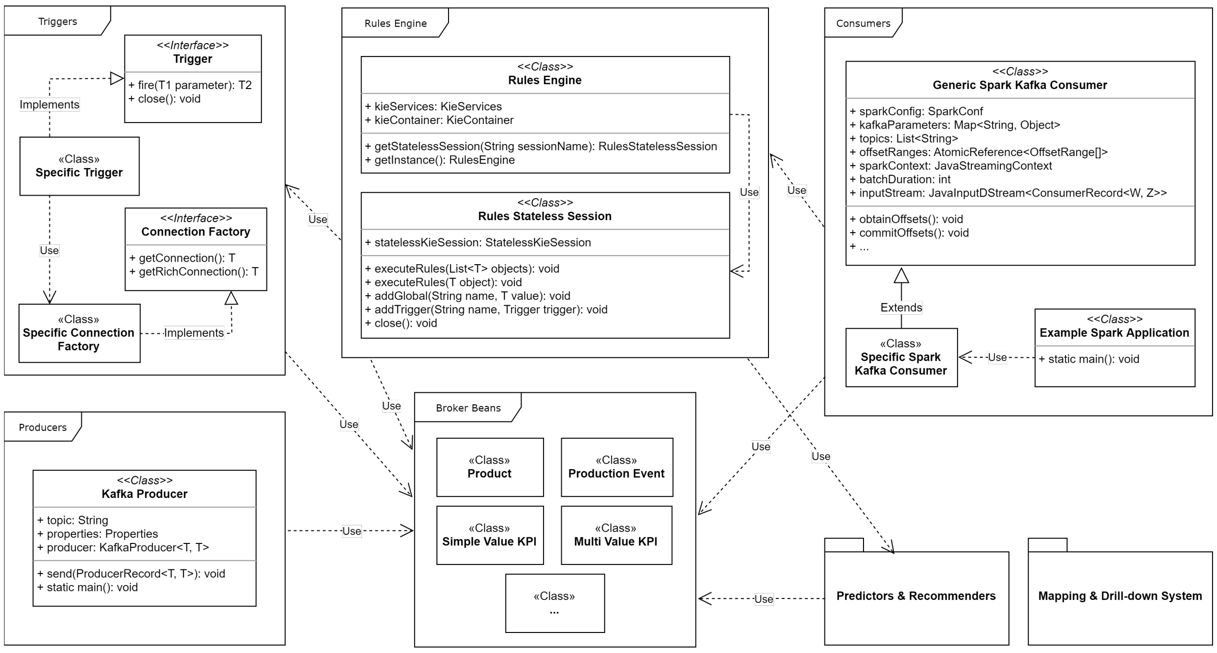

Triggers
<<Interface>>
Trigger
+ fire(T1 parameter): T2
+ close(): void
Implements
«Class»
Specific Trigger
Use
<<Interface>>
Connection Factory
+ getConnection(): T
+ getRichConnection(): T
«Class»
Specific Connection Factory
Implements
Rules Engine
<<Class>>
Rules Engine
+ kieServices: KieServices
+ kieContainer: KieContainer
+ getStatelessSession(String sessionName): RulesStatelessSession
+ getInstance(): RulesEngine
<<Class>>
Rules Stateless Session
+ statelessKieSession: StatelessKieSession
+ executeRules(List<T> objects): void
+ executeRules(T object): void
+ addGlobal(String name, T value): void
+ addTrigger(String name, Trigger trigger): void
+ close(): void
Consumers
<<Class>>
Generic Spark Kafka Consumer
+ sparkConfig: SparkConf
+ kafkaParameters: Map<String, Object>
+ topics: List<String>
+ offsetRanges: AtomicReference<OffsetRange[]>
+ sparkContext: JavaStreamingContext
+ batchDuration: int
+ inputStream: JavaInputDStream<ConsumerRecord<W, Z>>
+ obtainOffsets(): void
+ commitOffsets(): void
+ ...
Extends
«Class»
Specific Spark Kafka Consumer
<<Class>>
Example Spark Application
+ static main(): void
Use
Producers
<<Class>>
Kafka Producer
+ topic: String
+ properties: Properties
+ producer: KafkaProducer<T, T>
+ send(ProducerRecord<T, T>): void
+ static main(): void
Broker Beans
«Class»
Product
«Class»
Production Event
«Class»
Simple Value KPI
«Class»
Multi Value KPI
«Class»
...
Predictors & Recommenders
Mapping & Drill-down System

Figure 3-2. 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 3-2 due to the fact that their development is not completed, but they are seen as relevant design elements of the top-level project, reason why they are presented and described in this work.

3.4 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 3-3.

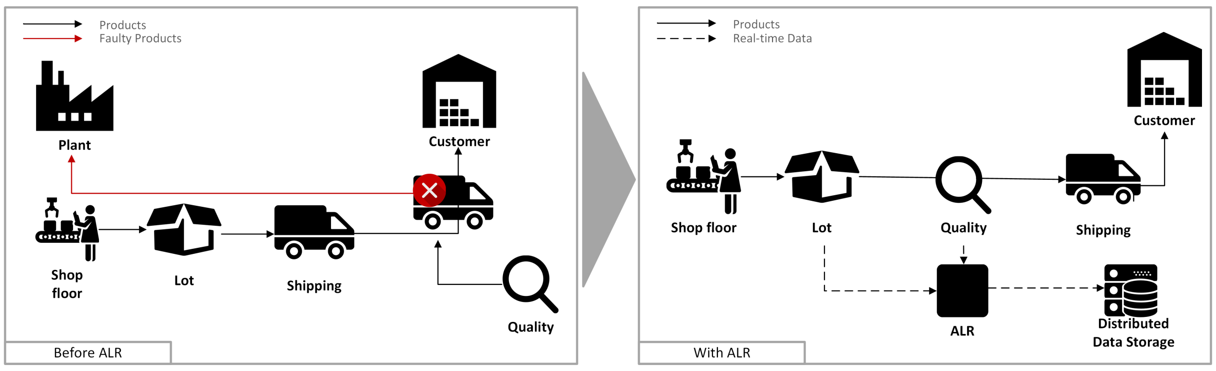


Figure 3-3. Quality control before and after the ALR implementation

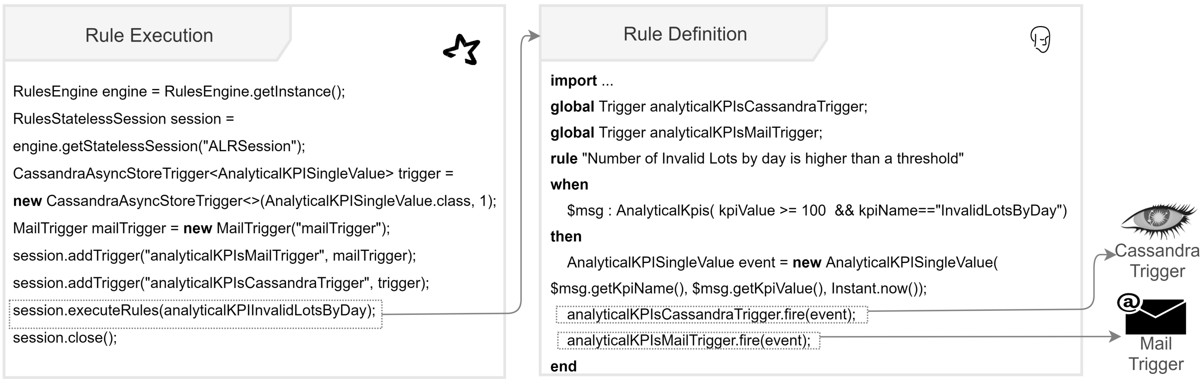
Focusing on the description of the demonstration case, the system implemented in this context is entirely based on the architecture depicted in Figure 3-1. As previously mentioned, the data comes from Bosch, thus it is important to implement some privacy policies. To accomplish this, the developed *Kafka Producer* uses a subset of the original ALR data (represented as an *IoT Gateway* in Figure 3-1), shuffling the data and updating its temporal fields before providing this data to *Kafka Consumers.* At this point, the timeframe used for the introduction of the events into the system was one message per second. This choice was made in this demonstration case just to test the system performance once 1 second is far less the actual time needed for the production and completion of a lot with several products. 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 (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 3-4 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 3-4 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*.



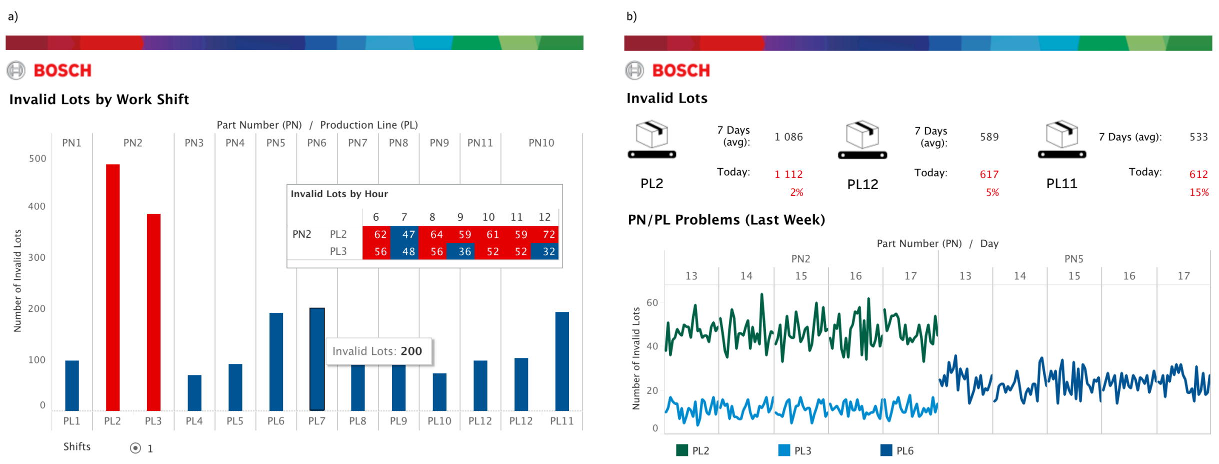
```
RulesEngine engine = RulesEngine.getInstance();
RulesStatelessSession session =
engine.getStatelessSession("ALRSession");
CassandraAsyncStoreTrigger<AnalyticalKPISingleValue> trigger =
new CassandraAsyncStoreTrigger<>(AnalyticalKPISingleValue.class, 1);
MailTrigger mailTrigger = new MailTrigger("mailTrigger");
session.addTrigger("analyticalKPIsMailTrigger", mailTrigger);
session.addTrigger("analyticalKPIsCassandraTrigger", trigger);
session.executeRules(analyticalKPIInvalidLotsByDay);
session.close();
```

```
import ...
global Trigger analyticalKPIsCassandraTrigger;
global Trigger analyticalKPIsMailTrigger;
rule "Number of Invalid Lots by day is higher than a threshold"
when
   $msg : AnalyticalKpis( kpiValue >= 100  && kpiName=="InvalidLotsByDay")
then
   AnalyticalKPISingleValue event = new AnalyticalKPISingleValue(
$msg.getKpiName(), $msg.getKpiValue(), Instant.now());
   analyticalKPIsCassandraTrigger.fire(event);
   analyticalKPIsMailTrigger.fire(event);
end
```

Figure 3-4. Example of the rule's 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 3-1). 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 3-4, 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 3-5) 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.



Invalid Lots by Hour

		6	7	8	9	10	11	12
PN2	PL2	62	47	64	59	61	59	72
	PL3	56	48	56	36	52	52	32

	7 Days (avg):	Today:	
PL2	1 086	1 112	2%
PL12	589	617	5%
PL11	533	612	15%

Figure 3-5. Example of operational and tactical dashboards

Taking this into consideration, in section a) of Figure 3-5 is showed 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, in section b) of Figure 3-5 it is presented 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.

3.5 Chapter Summary

This chapter 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 chapter 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 chapter 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.

Chapter 4. The Inspection and Logging System for the Intelligent Event Broker

This chapter[9] presents the data model for the Inspection and Logging System for the Monitoring System of the *IEB* (mentioned in Chapter 2 and on the 6th objective of this thesis). This chapter presents the related work identified in the literature; the design rules for the identification of the *IEB* metadata that should be collected by the system; a first version of the architecture for the inspection of the code and logging of the data; and a demonstration case highlighting how the inspection component of the system can be implemented, using the Bosch Portugal context.

4.1 Introduction

Nowadays, several industries are pursuing the adoption of Big Data and Real-time concepts in their enterprises. This need arose, for example, from the current technological evolution that results on a huge amount of data being produced every day by various types of machines inside the shop floor. Once the data is available, the challenge is how to use it to improve the organizations' performance by acting intelligently and without need to wait for human analysis and approvals. The Complex Event Processing (CEP) concept has existed since the 90s, always linked to the need to process various events in a real-time fashion. Nowadays, CEP is being integrated into Big Data contexts due to the need to process events resulting from real-time data streams that are more frequently available in the organizations.

The *Intelligent Event Broker (IEB)* is proposed in Chapter 3 and Andrade et al. (2019) as an innovative system that integrates the CEP and Big Data concepts using a Rules Engine embedded into Spark[10]. The

[9] This chapter integrates the content of the research paper "An Inspection and Logging System for Complex Event Processing in Bosch's Industry 4.0 Movement", published at The European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS'2020) (Andrade, Cardoso, Costa, & Santos, 2020). The contributions of Carina Andrade for this work were focused on the design of the *IEB* Mapping and Drill-down System architecture and the definition of the design rules to be applied for the inspection of the *IEB* system, as well as the development of the demonstration case to validate those rules. **Acknowledgements.** This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020, the Doctoral scholarship PD/BDE/135101/2017 and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 039479; Funding Reference: POCI-01-0247-FEDER-039479].

[10] https://spark.apache.org/

architecture considers the existence of several types of data sources and a component (*Producers*) dedicated to the standardization of the connection to all of them. The events arriving at the system are serialized into classes representing the business entities (*Broker Beans*) that will be subscribed by the *Event Processor* (Spark *Consumers*). This last component can send event data to be aggregated for further Key Performance Indicators (KPIs) calculation in the *Event Aggregator* (supported by Druid[11]). *Consumers* are also directly related to the *Rules Engine* (implemented in Drools[12]) where all the business requirements are defined as strategical, tactical, or operational *Rules*. These three types of *Rules* are translated at runtime by the *Consumers*. The *Triggers* component represents the connection to all the *Destination Systems*, and they will perform certain actions based on the results of the rules verification (e.g., stop a production machine if the last three products registered a failure). The *Predictors and Recommenders* is the component responsible for the application of previously trained Machine Learning models, which are stored in the *Lake of Machine Learning Models*.

Furthermore, it is considered that this kind of system needs closer and rich monitoring capabilities. In this context, a *Mapping and Drill-down System* was previously included in the *IEB* architecture, considering a *Graph Database* and a *Web Visualization Platform* to perform the system monitorization. This *Graph Database* will store the data related to the *IEB* codebase that is continuously and automatically inferred, as well as the relationships between them and the logs from the system components that are continuously running. The collected data will be analyzed in a *Web Visualization Platform* already proposed in Rebelo, Andrade, Costa, & Santos (2019). Therefore, this work aims to detail this system by proposing an architecture and a set of design rules that will ensure the adequate data collection to feed the *Mapping and Drill-down System*, ensuring the efficient inspection and logging of the *IEB*.

This chapter is structured as follows: Section 4.2 presents the related work identified in the literature; section 4.3 presents the inspection and logging system architecture and a set of design rules; section 4.4

[11] https://druid.apache.org/

[12] https://www.drools.org/

presents the demonstration case to highlight how the inspection component of the system was implemented, using the Bosch Portugal context; section 4.5 summarizes the presented work.

4.2 Related Work

Being this a very specific topic inside what is already a very specific research context (i.e., CEP on Big Data contexts), there is a lack of relevant literature and related works. Regarding the existence of systems that integrate the CEP concept in Big Data contexts, few works were found in the literature. The work of Hadar (2016) proposes an architecture (BiDCEP) for a system that integrates the CEP capabilities in the Big Data world. For that, the authors idealized a mixed Big Data Streaming architecture based on the recognized Lambda and Kappa architectures for Big Data. This proposal is summarized as being the extension of these architectures with components that represent the CEP system. The work of Ioannis Flouris et al. (2016) presents a prototype named FERARI that aims to process a large number of event streams in a multi-cloud environment. Their proposal is based on four components, each one with distinct goals: a web-based graphical user interface to define CEP concepts; a component to plan the latency and communication between the instances of the inter-cloud; a component responsible for the events processing and a web-based dashboard with reports regarding the processed events. Another architecture to integrate CEP and Big Data using only open source technologies is discussed in Jha, Jha, O'Brien, & Singh (2016), using an electronic coupon distribution centre as a demonstration case and giving focus to the technologies selection (uses Apache Kafka as a message broker, HDFS to store the data and a CEP system based on If-Then-Else rules to process the data).

During the analysis of the related work, when looking for the need of systems to monitor a CEP in Big Data contexts, works only revealed the possible use of CEPs to monitor other systems (Jayan & Rajan, 2014; F. Nguyen & Pitner, 2012). In this context, proper logging was considered since logs are widely used to indicate the state of the system at runtime, providing the details of the transactions that occur and containing useful information (e.g., name, date, and time of the occurrence (H. T. C. Nguyen, Lee, Kim, Ko, & Comuzzi, 2019)), which helps to understand the behaviour of the system. The work of Gupta (2003) mentions that a log must be recorded in an orderly and controlled manner so that it is human-readable. Nevertheless, its usefulness depends on how the log is applied, proposing much more than debug information and being of considerable value when analyzing the performance of an application (Gupta, 2003).

Regarding the monitorization of this kind of systems, this concern was only identified in the FERARI project, with a dashboard component that provides reports about some metadata of the system (e.g., daily events or for the last 4 hours) (Ćurin et al., 2016), being the focus on the data flows of the system and apparently leaving aside the drill-down into the insights and behaviour from the individual components of the system.

4.3 The Inspection and Logging System as a IEB Mapping and Drill-down Feeder

Considering the complexity that can arise with the evolution of a system like the *IEB* proposed in Chapter 3, especially when running it in industrial contexts, a dedicated system must work in parallel to guarantee the constant and long-term monitoring of the *IEB's* daily operations, ensuring its sustainable and controlled growth.

Taking this into consideration, the *Mapping and Drill-down System* was included in the *IEB* architecture and considers the implementation of two components: i) a *Graph Database* (considered the most adequate database due to the need to deal with the constant evolution of the *IEB* and its potential growth when running with several subjects simultaneously); and, ii) a *Web Visualization Platform* to enable the drill-down over the data, as discussed in (Rebelo et al., 2019). This work is focused on the design of the system that will allow the fueling of the *Graph Database*, to operate as the data source for the *Web Visualization Platform*, allowing the *IEB* stakeholders and operators to establish relevant relationships and drill-down operations into several occurrences within the system and its components.

4.3.1 System Architecture

As briefly highlighted in Chapter 3 and Andrade et al. (2019), the inspection and logging system architecture present here must ensure the following goals: **G1)** the proper analysis and indexing of the *IEB* codebase; **G2)** appropriate runtime logging mechanisms, to guarantee the storage of all the relevant data about the system components that are constantly working; **G3)** the analysis of the system functionality, recording what happened and when; and, **G4)** the analysis of the system performance (e.g., how many events were produced in *Producer X*, or consumed by *Consumer Y*). To ensure that all these defined goals are considered, the system architecture presented in Figure 4-1 is divided into two parts:

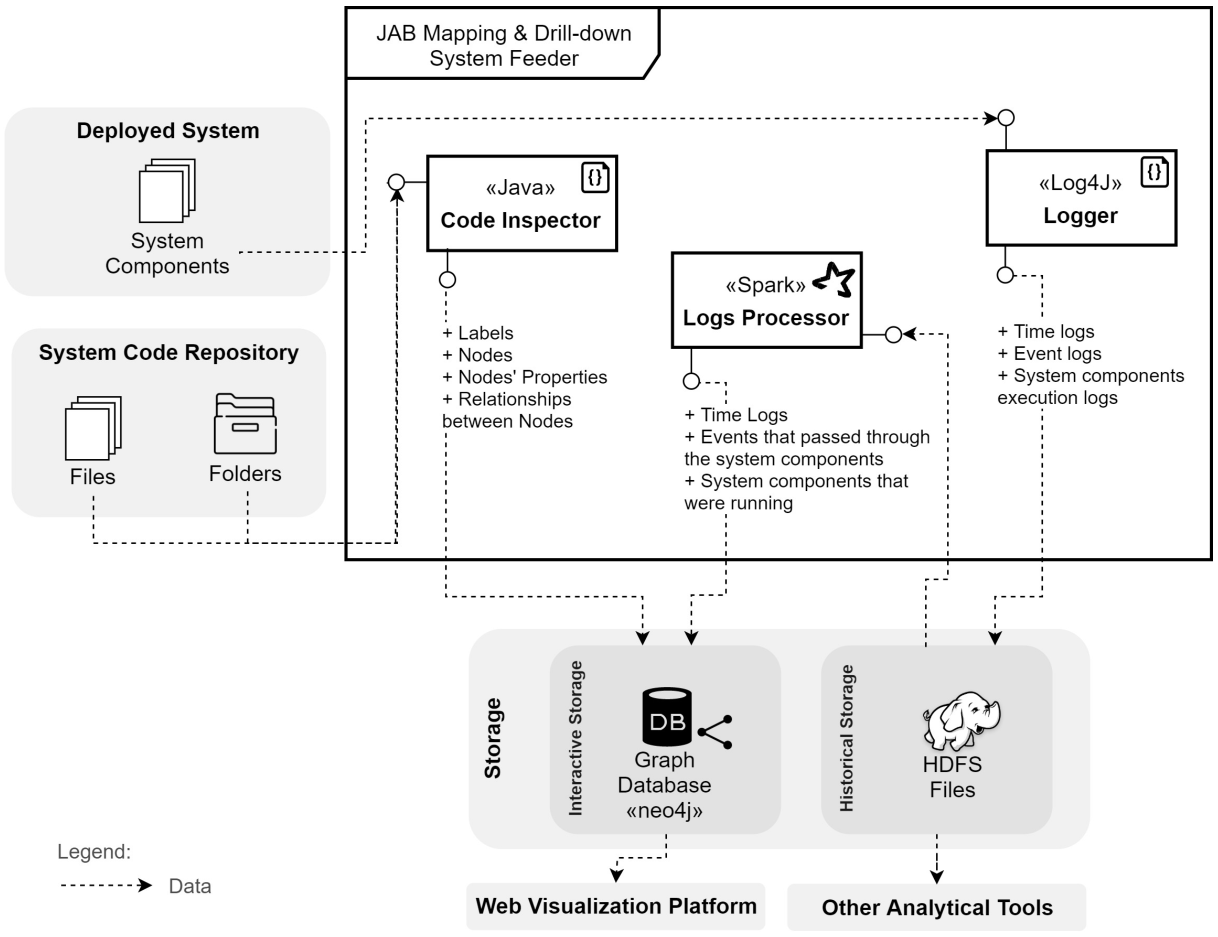


Figure 4-1. IEB Mapping and Drill-down system feeder architecture

1) The first part is dedicated to code inspection. For that, the *IEB* codebase (*System Code Repository*) is used as the source for analysis, exploring the folders and files that compose the system and using the *Code Inspector* component to collect data to feed the *Graph Database* (e.g., collect data to create graph labels, graph nodes, graph nodes' properties and relationships between graph nodes). To collect all the relevant data that will allow the definition of the Graph Data Model (GDM), a set of design rules were defined and are presented in subsection 4.3.2. The implementation of these rules

will guarantee the creation of the GDM regardless of how the system is implemented and addressing the previously presented goal **G1)**;

2) The second part is related to logging mechanisms. Here, the *IEB* components already implemented are considered as data sources to provide useful data when they are running. The *Logger* component should be embedded in the *IEB* components, logging the time, the events flowing through the system, and the system components execution. The time logs are the key point since time will be a property in the relationships between graph nodes, representing when something happened in the system. Secondary storage is proposed for historical and analytical purposes due to the dimension that the *Graph Database* can achieve.

   a) In the *Historical Storage*, all the raw data history will be available for analytical purposes, when ad hoc queries involving the use of complex interactions and calculations is needed. Here, analytical tools like Hive[13] or Spark can be used to explore the data in the Hadoop Distributed File System (HDFS), and data exploration tools like Zeppelin[14] or Tableau[15] can be used to interact visually with all the raw historical data.

   b) In *Interactive Storage*, a graph database has the most recent data to drill-down over it and take advantage of the analysis that can be done on graphs. The time frame defined for the data stored in the graph database depends on three factors: i) the business requirements; ii) the amount of data generated by the system; and, iii) the capacity of the neo4j infrastructure. The *Logs Processor* component will pick up the data from the *Historical Storage* (at predefined periods) to load it into the graph database (*Interactive Storage*). This will guarantee the achievement of the previously presented goals **G2), G3)** and **G4)**.

The architecture presented in Figure 4-1 already proposes technologies that can be used for each component (e.g., Spark for the *Logs Processor* component), considering the technological stack already

[13] https://hive.apache.org/

[14] https://zeppelin.apache.org/

[15] https://www.tableau.com/

proposed in Chapter 3 for the remaining system. Regarding the *Logger* component, Log4j[16] was proposed considering that is an open-source structure, flexible, written in Java and currently commonly used (Dickey et al., 2011). However, practitioners are free to choose other technologies for their specific implementation of a similar system to monitor their CEP system in Big Data contexts, as the conceptual proposal still holds true.

4.3.2 Design Rules for the Code Inspector Component

Related to the repository code inspection, several design rules are defined to guarantee that all the relevant *IEB* components are identified and tracked appropriately, as well as their relationships. These design rules take into consideration that the result of the code inspection should be a GDM to be implemented in a graph database. Once these design rules are implemented in a piece of software dedicated to inspecting the *IEB*, in any programming language practitioners choose to (ours is in Java due to the codebase of the *IEB*), all the relevant inspection data will be captured to create the graph (namely the inspection part of the graph) for the *Mapping and Drill-down System*. The defined design rules (DR) are presented as follow:

DR1: The *IEB* components are the 1st level labels. This will guarantee that all the implemented components from the *IEB* (mentioned in section 4.1) are categorized by their type of component in the architecture, facilitating the future exploration of the graph database (e.g., *Producers*, *Rules* or *Triggers*).

DR2: The folders' name, where the components were found, are 2nd level labels, if it defines a specific subject for the *IEB*. Considering the variety of subjects that can be implemented in the *IEB*, a clear separation by folders should be made during the implementation. The collection of this information will enable the differentiation between the several topics implemented in the system (e.g., "/producers/alr" or "/consumers/shop-floor-incidents").

[16] http://logging.apache.org/log4j/2.x/

DR3: The files' name that reflects the implementation of *IEB* components are the names of the graph nodes. The graph nodes will be the several instantiations of the system components already implemented for the various *IEB* contexts (e.g., *Producers* or *Rules* for a subject).

DR4: The labels defined in DR1 and DR2 must be associated to the graph nodes identified in DR3. This will guarantee that all the created graph nodes will have at least one label associated, identifying the system component and (in some cases) the subject related to their implementation.

DR5: In the *Broker Beans*' files, their variables and data types are their graph nodes' properties. Considering the importance of this component when representing the data that will flow throughout the system, it is relevant that the *Broker Beans* variables and respective data types are collected.

DR6: In the *Broker Beans*' files, if the variable data type is another *Broker Bean*, a relationship between the first identified *Broker Bean* graph node (BB1) and the one identified in the variable data type (BB2) should be created as "BB1 composed_of BB2". Since the *Broker Beans* represent the business entities flowing in the system and, in some cases, a business entity can be composed of other business entities, this relationship should be identified.

DR7: In the *IEB* components' files, a relationship between graph nodes should be created when Inheritance or Implementation relationships are identified. For some components (e.g., *Producers* or *Consumers*), the collection of data representing the Inheritance and Implementation between the files that define them is relevant and must be ensured.

DR8: For any system component that instantiates another one, a relationship between the component (Cp1) and the one identified as being instantiated (Cp2) should be created as "Cp1 instantiates Cp2". This step will ensure that all the relationships between the components are stored to generate knowledge about the interaction of the components.

DR9: For the *Consumers*, it should be identified if they verify certain *Rules*, creating a relationship between the *Consumer* (C1) and the Rules Session that is executed (RS1) as "C1 runs RS1" and which *Rule* (Ru1) is verified as "C1 verifies Ru1". With this design rule, it is guaranteed that the *Consumers* running the Rules Session (a set of *Rules* within all the defined *Rules*) and therefore verifying certain *Rules*, are identified, and properly stored in the graph.

DR10: For each *Consumer*, it should be identified if it queries or stores data in the *Event Aggregator* component. A relationship between the *Consumer* (C1) and the used *Event Aggregator* (EA1) should be created as "C1 queries EA1". Moreover, a relationship between the *Consumer* and the *Event Aggregator* where it stores new data (EA2) should be created as "C1 stores_data_in EA2". This design rule identifies an interaction between a *Consumer* and the *Event Aggregator* in both directions: querying and storing data on it.

DR11: For the identified *Triggers*, it should be created a relationship between the *Trigger* (T1) and the *Destination System* (DS1) that will receive the data sent by the *Trigger* as "T1 propagates_data_to DS1". The actions defined in the *Triggers* can send data to different *Destination Systems* identified in the system architecture presented in Chapter 3, the reason why each *Trigger* should have a relationship with the graph node that represents the *Destination System* being used in that action.

DR12: Regarding the rules' repository, it should be collected the *Rule*'s name and the *Trigger* fired by the *Rule*, as well as the *Broker Beans* used for the *Rule* verification and to trigger the action. The *Rule*'s names should be saved as graph name nodes and the *Trigger* and *Broker Beans* identified are used for design rules DR13 and DR14. Depending on the Rules Engine being used, different ways for the rules definition can exist (i.e., Drools as a way of defining rules, while other business rules systems may have others). Nevertheless, the *Rules* must be identified and stored in the graph, as well as the *Triggers* and the *Broker Beans* used by them.

DR13: For the *Rules* identified in DR12, it should be created a relationship between the *Rule* graph node (Ru1) and the *Broker Bean* (BB1) used for the *Rule* verification as "Ru1 uses BB1". Moreover, it should be created a relationship between the *Rule* and the graph node that represents the *Trigger* (T1) fired by the *Rule* as "Ru1 fires T1". These relationships will ensure the tracking of which *Broker Beans* are used by the *Rules* verification and which *Triggers* are fired by which *Rule*.

DR14: For the *Triggers* identified in DR12, it should be created a relationship between the *Trigger* graph node (T1) and the *Broker Beans* (BB1) used to take any action as "T1 uses_to_trigger BB1". It is necessary to identify which specific *Broker Beans* are used for the system's actions ensuring the identification of the data that are propagated to the *Destination Systems* (e.g., IoT Gateways or a database to feed Analytical Applications, as identified in the *IEB* architecture (Chapter 3 and Andrade et al. (2019)).

4.4 Demonstration Case

In this section, it is presented a demonstration case for the *Code Inspector* component of the architecture proposed in section 4.3. First, it is presented a flowchart that reflects the implementation of the set of design rules defined in subsection 4.3.2, using the *IEB* implementation in the Bosch Portugal ALR[17] data context. Then, for the obtained flowchart, a part of the GDM is demonstrated and explained, resulting from the implementation of the steps in the flowchart. Although the *Logger* and *Logs Processor* components are already being developed, for this chapter, this component is defined at the conceptual level, and its demonstration is identified as future work.

4.4.1 Code Inspector Flowchart

The flowchart (Figure 4-2) that represents the implementation of the design rules defined in subsection 4.3.2 is based on the *IEB* system already presented in Chapter 3 and Andrade et al. (2019). To properly interpret this diagram, remember that the *IEB* was implemented using Java and Drools (this last one, as Rules Engine).

In this context, each package represents the implementation of one of the *IEB* components and to save this data, the *Code Inspector* seeks throughout the packages to analyze the implemented code, storing the package name as 1st level label (**DR1**). When inside a package, it searches for *.java* and *.drl* files (Drools files) and for each file, its directory name is saved (if it does not exist yet) as 2nd level label (**DR2**). After that, the file name is saved as the graph node, representing the class that is part of the system implementation (**DR3**). The identified labels are then linked to the created node (**DR4**). All these steps

[17] A system that verifies if a lot can be shipped to the customer.

are executed until there are no more packages and no more *.java* or *.drl* files to identify. This will guarantee that the graph nodes and labels needed for the creation of the relationships already exist.

With all the graph nodes already created, the *Code Inspector* starts exploring the files again in the first package. If the selected file is from the *Broker Beans* package, the private variables names and properties are saved as node properties for the graph node previously defined with the same name as the file name being analyzed (**DR5**). If some of the variable's types represent other nodes identified in the *Broker Beans* package, a relationship between the file/graph node being analyzed and the graph node representing the variable type is saved as shown in Figure 4-2 (**DR6** - *composed_of* relationship).

For the rest of the *.java* files identified in the packages, if they include a string *"extends"* or *"implements"* followed by another graph name node previously identified, a relationship between the file/graph node being analyzed and the one identified after the mentioned strings are saved as identified in the flowchart (**DR7** - *extends* and *implements* relationships). The string *"new"* followed by another graph name node previously identified will allow the identification of instantiations being carried out by the file under analysis. A relationship is saved as a file/graph node that *instantiates* another graph node (**DR8**). The same happens for the identification of which *Consumers* run the *RulesSession*. In this case, searching for the string "*getStatelessSession*" and saving a relationship as the graph node representing the file being analyzed *runs* the "*ruleSessionName*" (the *getStatelessSession* parameter) identified.

Furthermore, the type of the parameter from the "*executeRules"* method will allow the identification of the *Rules* verified in that session (all the *Rules* waiting for a specific parameter type inside a session will be verified by the *Consumer* - **DR9**). Regarding the *Event Aggregator* component, this one can be queried, and the data can be stored on it. If the file contains the string *"querying/"* or the *"consumerType"* equals *"Druid"*, *Event Aggregator* is saved as 1st level label.

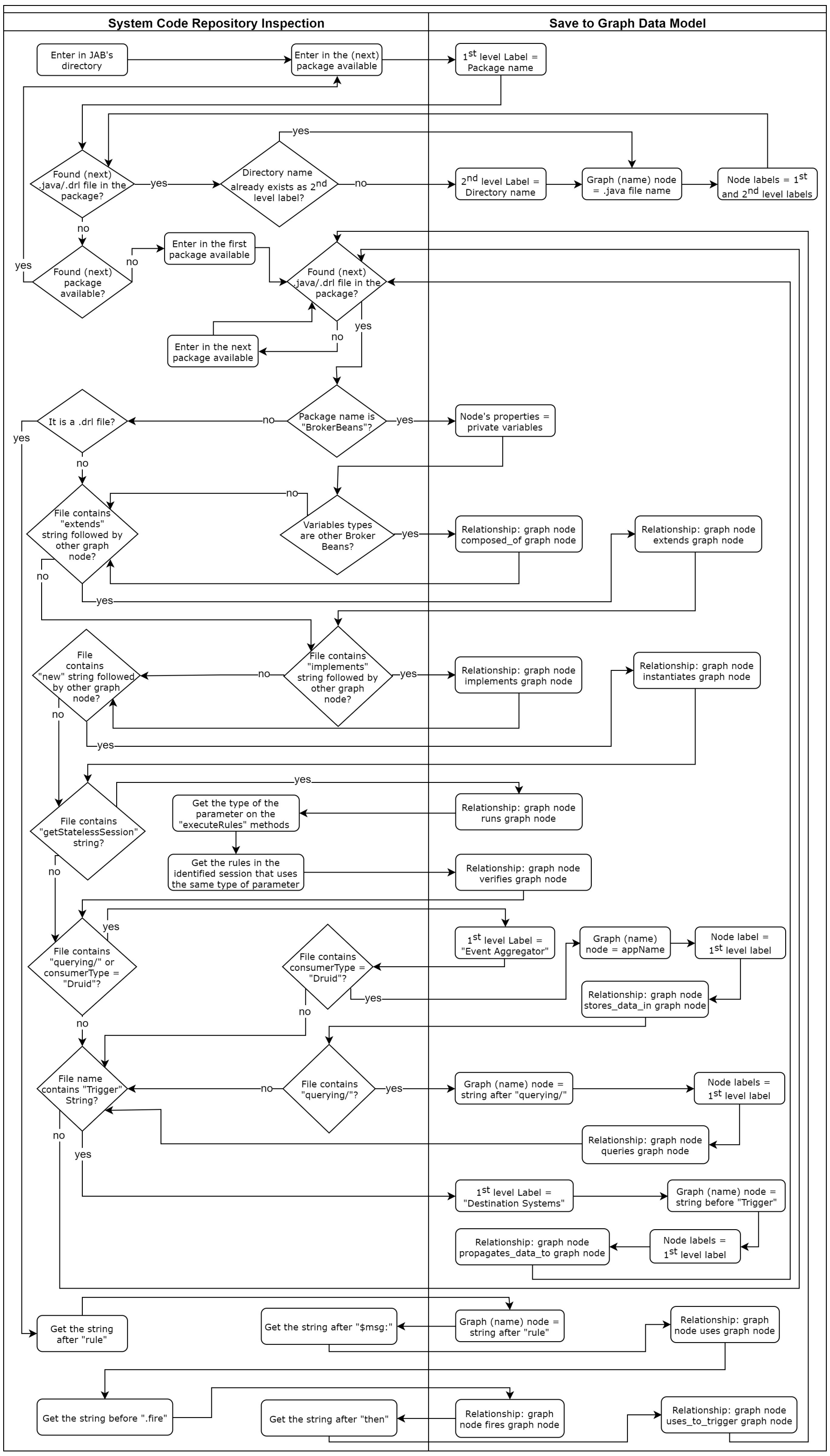

System Code Repository Inspection
Save to Graph Data Model
Enter in JAB's directory
Enter in the (next) package available
1st level Label = Package name
Found (next) java/.drl file in the package?
Directory name already exists as 2nd level label?
2nd level Label = Directory name
Graph (name) node = .java file name
Node labels = 1st and 2nd level labels
Found (next) package available?
Enter in the first package available
Found (next) java/.drl file in the package?
Enter in the next package available
It is a .drl file?
Package name is "BrokerBeans"?
Node's properties = private variables
File contains "extends" string followed by other graph node?
Variables types are other Broker Beans?
Relationship: graph node composed_of graph node
Relationship: graph node extends graph node
File contains "new" string followed by other graph node?
File contains "implements" string followed by other graph node?
Relationship: graph node implements graph node
Relationship: graph node instantiates graph node
File contains "getStatelessSession" string?
Get the type of the parameter on the "executeRules" methods
Get the rules in the identified session that uses the same type of parameter
Relationship: graph node runs graph node
Relationship: graph node verifies graph node
File contains "querying/" or consumerType = "Druid"?
File contains consumerType = "Druid"?
1st level Label = "Event Aggregator"
Graph (name) node = appName
Node label = 1st level label
Relationship: graph node stores_data_in graph node
File name contains "Trigger" String?
File contains "querying/"?
Graph (name) node = string after "querying/"
Node labels = 1st level label
Relationship: graph node queries graph node
1st level Label = "Destination Systems"
Graph (name) node = string before "Trigger"
Node labels = 1st level label
Relationship: graph node propagates_data_to graph node
Get the string after "rule"
Get the string after "$msg:"
Graph (name) node = string after "rule"
Relationship: graph node uses graph node
Get the string before ".fire"
Get the string after "then"
Relationship: graph node fires graph node
Relationship: graph node uses_to_trigger graph node
yes
no

Figure 4-2. Flowchart of the application of the design rules in the IEB

For the ones in which the *"consumerType"* equals *"Druid"*: i) the *"appName"* (that represents the *Event Aggregator*) is saved as graph name node; ii) the 1st level label is defined as a label for the created node; and, iii) a relationship is created as the node representing the file being analyzed *stores_data_in* node representing the *Event Aggregator*. The same happens when the *"querying/"* string is found, being the relationship defined as *queries* instead of *stores_data_in* (**DR10**). On the other hand, if the file name contains the *"Trigger"* string: i) *Destination Systems* is saved as 1st level label; ii) the name before *"Trigger"* is saved as graph name node; iii) the 1st level label is defined for the created node; and, iv) a relationship is created between the node representing the file being analyzed (*Trigger*) and the node created as being the *Destination System* ("*Trigger propagates_data_to Destination System*") (**DR11**).

Concerning the *Rules* defined in *.drl* files (Drools files), a file exploration process is executed to collect: i) the *Rules* that are in quotes after the *"rule"* string; ii) the *Broker Beans* used for the verification of the rules; iii) the *Trigger* to be fired by the rule; and, iv) the *Broker Bean* to be fired by the *Trigger*. With this data, are created: i) the graph node with the *Rule* name (**DR12**); ii) a relationship between the *Rule* graph node and the *Broker Bean* used for its verification (**DR13**); iii) a relationship between the *Rule* graph node and the *Trigger* that is fired (**DR13**); and, iv) a relationship between the *Trigger* and the *Broker Bean* that was flowing when the *Trigger* was fired (**DR14**).

With the definition of the flowchart, it is possible to understand that the design rules presented in subsection 4.3.2 are easily transformed into small tasks to be coded and applied to the implemented system. Although the system is implemented using Java and Drools, other technologies can be used if the main guidelines are followed, as the application of the design rules returns the data needed to be monitored.

4.4.2 Graph Data Model

In Figure 4-3, it can be seen the representation of the GDM that was obtained from the *Code Inspector* component to support the *Mapping and Drill-down System* (see subsection 4.3.2). Due to the difficulty of presenting here the whole GDM created during this work, this figure only shows an example of the relationships between the possible types of graph nodes. Nevertheless, the whole data model contains sixteen labels, more than fifty nodes and more than sixty relationships.

Before starting the explanation of the nodes (circles) and relationships (edges) of the GDM in Figure 4-3, it should be considered that the nodes' backgrounds represent the packages' name (1st level label mentioned in **DR1**), and the lines around the nodes represent the folders' names (2nd level label mentioned in **DR2**) as shown in the figure legend. **DR3** and **DR4** are explicit in the GDM since the nodes are clearly identified and all of them have a specific colour.

Considering this, the different types of graph nodes and relationships between them are described as follow:

- The *ALRProducer* extends a generic implementation of a producer developed using Apache Kafka *(KafkaEventProducer)*, which by itself implements the *Producer* interface (**DR7**). This aims to standardize the implementation of the *Producers*, creating a generic class and an interface that reflects the behaviour of every producer in the system.

- Three types of *Broker Beans* can be seen: i) *Broker Beans* representing the business entities that arrive at the system through the events (*ALR*, *Lot* and *PickCoordinate*); ii) *Broker Beans* representing the KPIs calculated by the system (*MultiValueKpi*); and, iii) *Broker Beans* created during the verification of the *Rules* to be later used by the *Triggers* (*LineEvent*). Regarding the *Broker Beans* that represent the business entities, these are instantiated by *Producers* and *Consumers* (**DR8**). They can also be composed of other *Broker Beans* (an *ALR Broker Bean* is composed of a *Lot Broker Bean* which subsequently is composed of a *PickCoordinate Broker Bean* - **DR6**). The KPIs *Broker Beans* (*MultiValueKpi*) are instantiated by *Consumers* with analytical purposes (*AlrAnalyticalKpisConsumer*) (**DR8**) and they are used by the *Rules* dedicated to the same analytical goals *(% Invalids Lots by day is bigger than a threshold*) (**DR13**). The third type of *Broker Beans* (*LineEvent*) is used by the *Triggers* that propagate them to the *Destination Systems* (**DR14**). For every *Broker Bean*, their variables and data types are identified as node's properties (**DR5**).

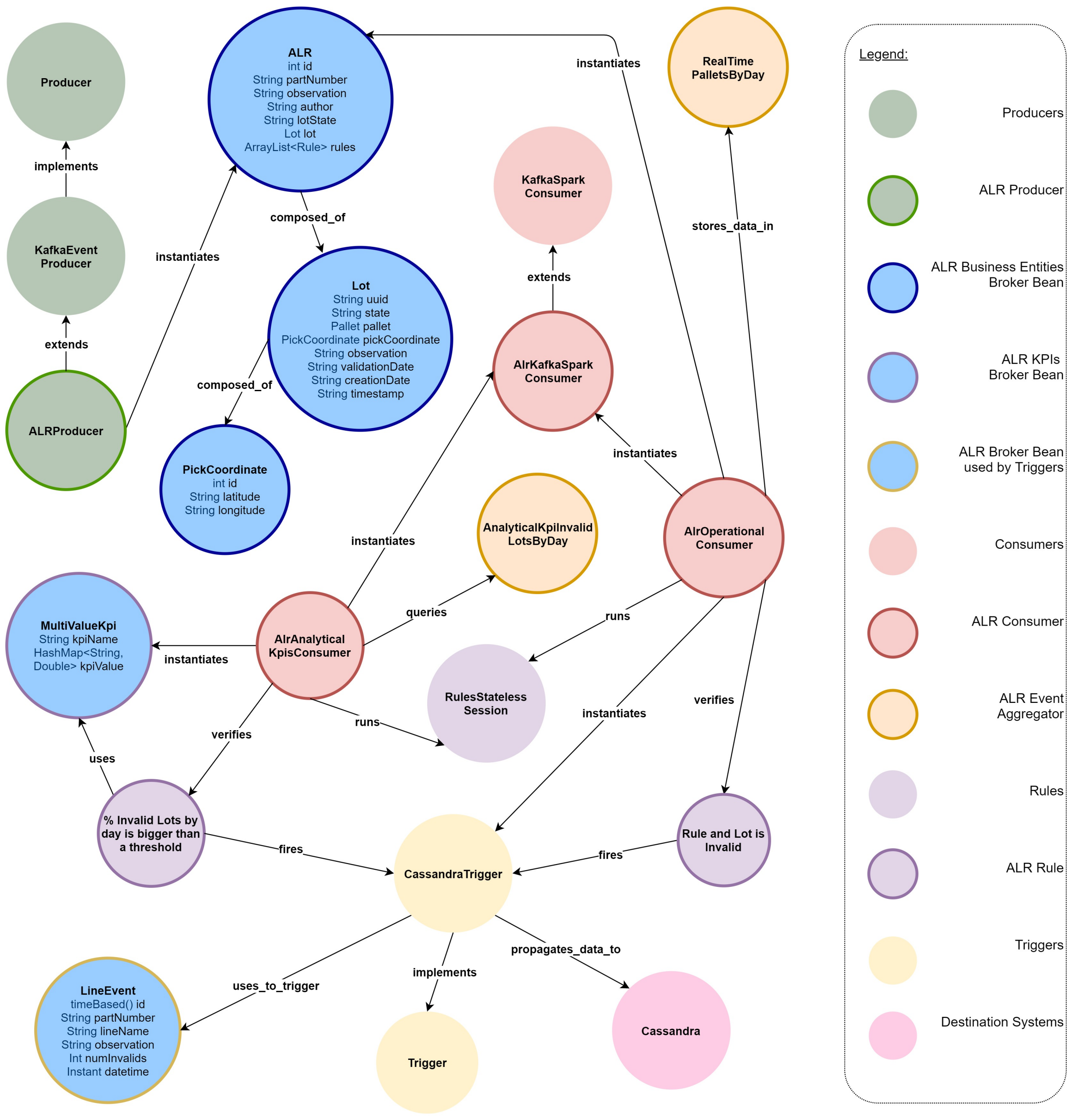


Figure 4-3. GDM resulting from the application of the set of proposed design rules

- The operational or analytical *Consumers* (*AlrOperationalConsumer* and *AlrAnalyticalKpisConsumer*) instantiate (**DR8**) a specific class (*AlrKafkaSparkConsumer*) that extends the *KafkaSparkConsumer* (**DR7**), being this a specific code design strategy in the *IEB*. Both of them run the *RulesStatelessSession* and verify the defined *Rules* in the GDM (**DR9**).

Furthermore, *Consumers* have two types of interactions with the *Event Aggregator* presented in the GDM (e.g., relationship *stores_data_in* between the *AlrOperationalConsumer* and the *RealTimePalletsByDay,* and relationship *queries* between the *AlrAnalyticalKpisConsumer* and the *AnalyticalKpiInvalidLotsByDay* - **DR10**). Finally, *Consumers* instantiate the *Triggers* (e.g., *CassandraTrigger*) that can be fired after the verification of the *Rules* (**DR8**).

- Two *Rules* are defined in the GDM *(% Invalids Lots by day is bigger than a threshold* and *Rule and Lot is Invalid* - **DR12**) with a relationship to the *Trigger* node reflecting that a *Rule fires* a *Trigger* (**DR13**).
- Regarding the *Triggers* component, the *CassandraTrigger* implements the *Trigger* interface (**DR7**) and *propagates_data_to* the *Cassandra Destination System* (**DR11**).

The detailed GDM (Figure 4-3) as well as the design rules identified in the description of the GDM, reflect the successful application of the design rules in the *IEB* system being demonstrated in the Industry 4.0 movement of Bosch Portugal. The codebase of the system was thoroughly analyzed by the authors, comparing it to the generated GDM. With this validation, it was possible to conclude that, as an initial prototype, the data needed to monitor the continuous growth of the system (e.g., new *Producers* or *Consumers*) is adequately identified and captured by the design rules proposed.

Regarding the *Logger* component, the experimental work in progress is currently focused on logging the execution of the *ALRProducer* and the *AlrOperationalConsumer*. The logs are being stored and processed as described in section 4.3 and the *Web Visualization Platform* (Rebelo et al., 2019) is used (here with a new design) for the data analysis. In the analysis presented in Figure 4-4, it can be seen the *ALRProducer*, the *AlrOperationalConsumer* and the *CassandraTrigger* nodes (green, red and yellow nodes) connected to all the *ALR Broker Beans* nodes (blue). In the future, we will work on extending the GDM presented here with the processed logging information from the *IEB* system.

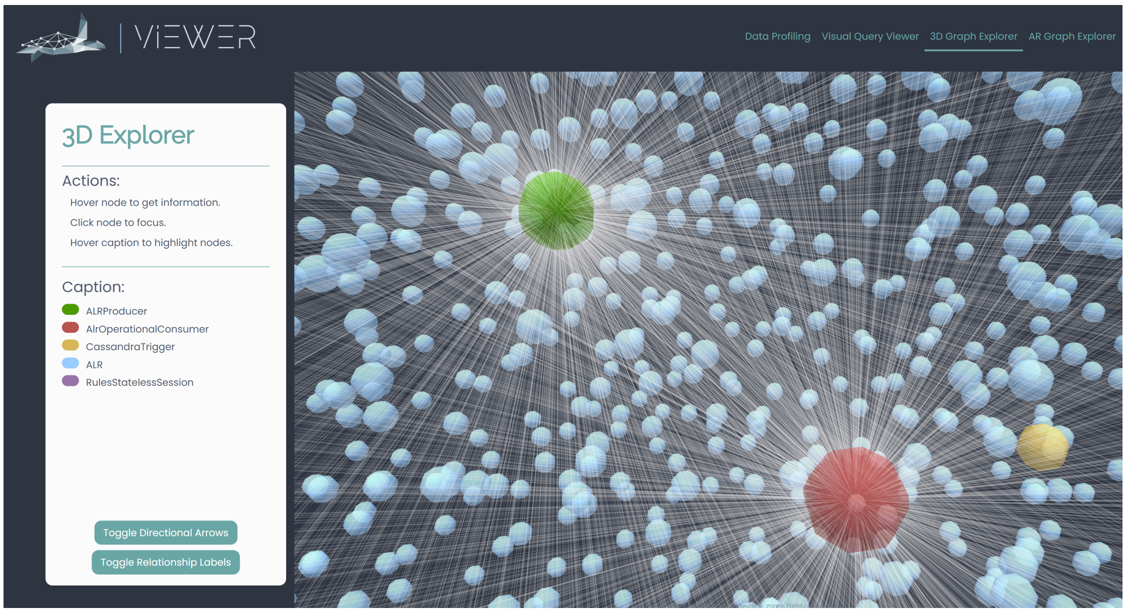


Figure 4-4. Data from the logger component presented in the 3D graph explorer of the IEB web visualization platform

4.5 Chapter Summary

Given the potential complexity of the *IEB* running in industrial contexts, we consider that adequate monitoring of this system is essential. This monitoring was conceptually considered in the *IEB* system (Chapter 3 and Andrade et al. (2019)) with the proposal of the *Mapping and Drill-down System* supported by graph storage and analysis technologies.

In this chapter, the architecture for the whole feeding system that will fuel the *Graph Database* was presented and discussed considering two main parts: i) the *Code Inspector*, and, ii) the *Logger*. This feeding system is responsible for generating the ever-growing graph mentioned above, addressing innovative monitoring capabilities for a CEP in Big Data contexts.

For the *Code Inspector* component, the main focus of this chapter, a set of design rules was defined to ensure that the *IEB* components are continuously and automatically inferred from the codebase, as well the relationships between them. The design rules were applied in a Bosch Portugal demonstration case using the ALR data, showing a flowchart on how to properly inspect the *IEB* codebase, and highlighting the results in a detailed GDM. The design rules were successfully applied to the *IEB* system returning all the useful graph data (labels, nodes, relationships) that is needed to feed the *IEB Mapping and Drill-down System*.

Despite the focus on the *IEB* system being demonstrated in the Bosch Portugal context, we believe that the artefacts and insights provided here, which complement the ones in Andrade et al. (2019) and Rebelo et al. (2019), are conceptually, technically and technologically relevant for several researchers and practitioners in the area. No other system similar to the *IEB* considers the relevance of monitoring using consistent strategies that focus on the evolution and growth of the system when working in production contexts, being this a key point to guarantee the adequate maintenance of the system in many contexts like the industry 4.0 movement.

Chapter 5. The Monitoring System for the Intelligent Event Broker

This chapter[18] defines the design for the implementation of the Monitoring System (6th objective of this thesis, identified in Chapter 2), explaining how all the data identified as being relevant to Monitor the System can be collected, processed, stored, and then used in the Web Visualization Platform.

5.1 Introduction

Currently, there is a vast amount of data that is constantly being produced. For example, communications between different machines generate data at high velocity and there is a need for automated decision-making processes. The Complex Event Processing (CEP) concept allows applications to extract, understand and transmit valuable information to recognize potentially relevant situations.

In this sense, the huge amount of heterogeneous data constantly generated by a world of interconnected things and the need for more advanced decision-making processes have significantly increased the need to explore and use Big Data, demanding that these decisions should be made in real time, as one of the main challenges is to obtain value from real-time (streaming) data. Thus, it is essential to integrate CEP in the era of Big Data to create innovative architectures capable of processing a large volume of events in a simple, scalable, and integrated way.

To address this gap identified in the literature, we have proposed the Intelligent Event Broker (*IEB*) architecture (Chapter 3 and Andrade et al. (2019)). The *IEB* is a collection of several components that are integrated and validated to create a homogeneous streaming-oriented CEP system for Big Data contexts. Due to the complexity of the *IEB* (that can have several business areas integrated into the system with a vast amount of data and rules being processed), it was necessary to create a component that runs

[18] This chapter integrates the content of the research paper "Designing Monitoring Systems for Complex Event Processing in Big Data Contexts", published at The European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS'2021) (Andrade, Cardoso, Costa, & Santos, 2021). The contributions of Carina Andrade for this work were focused on detailing and defining the technological instantiation of the *IEB* Mapping and Drill-down System architecture, as well as ensuring the correct implementation of some of the design rules defined in the previously chapter and validating the results when implementing the demonstration case for the *IEB* Mapping and Drill-down system. **Acknowledgements.** This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020, the Doctoral scholarship PD/BDE/135101/2017 and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 039479; Funding Reference: POCI-01-0247-FEDER-039479].

in parallel to the main components of the system, ensuring the constant monitoring of the *IEB*'s daily operations. That specific component is called *Mapping and Drill-down System* and has as key components a graph database, proposed in Chapter 4 and Andrade, Cardoso, Costa, & Santos (2020), and a Web Visualization Platform, proposed in Chapter 6 and Rebelo et al. (2019).

To clarify the incremental valuable contribution of this work to the scientific and technical community, it is relevant to highlight that although the *Mapping and Drill-down System* is briefly mentioned in Chapter 3 and Andrade et al. (2019) and the Logger component of the same system was not detailed nor developed in that paper, being now necessary to propose a detailed architecture and demonstration case for this specific component, due to the complexity of ensuring a logging/monitoring system for a CEP system in Big Data contexts, in this case, specifically for the *IEB*, but with enough generalization to make it applicable to other CEP systems in Big Data contexts. This is seen as the main contribution of this work, allowing researchers and practitioners to take advantage of all the logs generated by the *IEB* (or apply the same ideas to any similar system they intend to propose or implement) and to exploit all the resulting information, in order to make more assertive decisions about the daily operations of a CEP system in Big Data contexts. Therefore, the main problem that this work is aiming to tackle is the adequate monitoring of CEP systems in Big Data environments, by proposing a monitoring system that works in parallel with the *IEB*, ensuring its justifiable and controlled growth. The data collected should indicate what happened, and when it happened, and should support an analysis of the system's performance.

In this context, we can consider a production line having various production machines, working every day. Each production machine has a specific function in the final product assembly process and after finishing the steps of a specific machine, the product flows to the next one. If we guarantee that the *IEB* system receives the data from each machine in a real-time context and we have rules defining what is a good result for a parameter of the product assembly, we can track the product for the next machine or block it for quality control. With that, we are preventing more components from being spent unnecessarily once some parameter does not have the expected value (and was detected prematurely).

Regarding our research process, the Design Science Research Methodology for Information System (Peffers et al., 2007) was followed since our main goal is to produce an IT artefact, namely a model (with its architecture and the supporting data model) for a logging and monitoring system for CEP in Big Data contexts. In this chapter, we also present a demonstration case to access the efficacy of the proposed solution. This is very important to validate this work and to show that we can effectively use the proposed model to monitor the *IEB* and hopefully contribute with an approach that other practitioners and researchers can use to propose similar systems.

This chapter is structured as follows: section 5.1 presents an introduction to the topic; section 5.2 gives a brief overview about the related work identified in the literature; section 5.3 proposes the system architecture for the *Mapping and Drill-down System*; section 5.4 describes a demonstration case focused on a prototype of the system; and section 5.5 summarizes the presented work.

5.2 Related Work

Ensuring adequate logging mechanisms in a system allows us to have an overview of the system behavior in production. Logs can be seen as messages collected from a specific point of the system that can have different formats and purposes, as mentioned by Oliner, Ganapathi, & Xu (2012): i) performance – to optimize the system performance; ii) security – to detect security problems; iii) forecast – including logs in the forecasting models; and, iv) reports – to provide details about the users' profiles, for example.

In the Big Data era, systems tend to use more complex and dynamic components, some of them including distributed processing, communication between different networks or different data sources, among others. Logging Big Data Systems can help to ensure that each interaction between components will be monitored, to prevent or track failures (Miranskyy, Hamou-Lhadj, Cialini, & Larsson, 2016).

Regarding the integration of the CEP concept with the Big Data era, some works are aiming to move forward in this research field by proposing system architectures that integrate these two topics. The work of Hadar (2016) and the work of Ioannis Flouris et al. (2016) propose two distinct architectures that achieve this purpose but lacks considerations such as the one focused on this chapter: a monitoring system to prevent the uncontrolled growth of the main system (*IEB*), which can be caused by the inclusion of several business processes to be actively followed, checking business requirements and acting over the results of their verification. Besides the fact that this type of need is not considered in the few proposed architectures for CEP in Big Data, to the best of our knowledge, there is no available system architecture such as the one proposed in this work.

According to the literature, there seems to be a lack of contributions focusing on the logging and monitoring of CEP systems in Big Data environments, which is understandable since the merge of these two topics is also a novel area. However, there is existing literature that focuses on logging/monitoring strategies for traditional CEP systems: Lan, Shi, Wang, Zhang, & Jiang (2019) present a logging/monitoring system for CEP, focusing on IoT devices, in which the authors discuss the monitoring of the arrival and processing of events. The authors mention the need to process huge amounts of data with low latency. Jayan & Rajan (2014) also discuss a logging system for CEP that focuses on network security logs, which include operating system logs, network logs and security devices logs. Similar to Lan et al. (2019), the authors focus on the volume of the logs, mentioning the problem of having a huge amount of logs, which may affect the performance of the CEP system, reason why we believe that works studying the combination of Big Data technologies with CEP systems, and the adequate logging and monitoring strategies of those, result in significantly valuable contributions to the community.

Considering that the work presented in this chapter is proposing new components and capabilities that are related to previous work proposed in Chapter 3 and Chapter 4, and considering the complexity of the *IEB* System proposed and discussed in those previous works, an introduction of its several components is needed, which can be seen in Figure 5-1. Given the variety of data sources (*Source Systems*) and formats available in many organizational contexts, it can become difficult to collect and process all this data from the *IEB*. With this, Kafka (a distributed streaming platform) is used to collect, disseminate, and standardize events (*Producers* component).

The events are collected from its source system using Kafka Producers developed for this purpose. The *Producers*, after collecting the data, serialize them in the form of Broker Beans (simple classes that represent the business information) and produce the events by publishing them in a Kafka cluster (*Brokers* component). These topics containing the published events are subscribed by the *Event Processor* component (Kafka Consumers) which is continuously processing new events. The *Event*

Processor is therefore one of the most relevant components of the *IEB* and it acts in collaboration with the *Rules Engine* component, which is responsible for executing several business rules defined within the organizational context. The Rules Engine component establishes communication with three other components: i) *Rules* - business requirements are converted into strategic, tactical, and operational rules; ii) *Triggers* - represent the connections to the various *Destination Systems*, i.e., after the condition of a rule is evaluated as true, these *Triggers* can, for example, perform some action in IoT Gateways, send an e-mail or just send data for an analytical application; iii) *Predictors and Recommenders* - ensure the ability to interpret previously trained Machine Learning models and use them to predict occurrences and recommend actions. Moreover, the *Event Processor* output can also be submitted to the *Event Aggregator*, which is responsible for receiving the data, performing aggregations, and storing them to calculate the relevant Key Performance Indicators (KPIs).

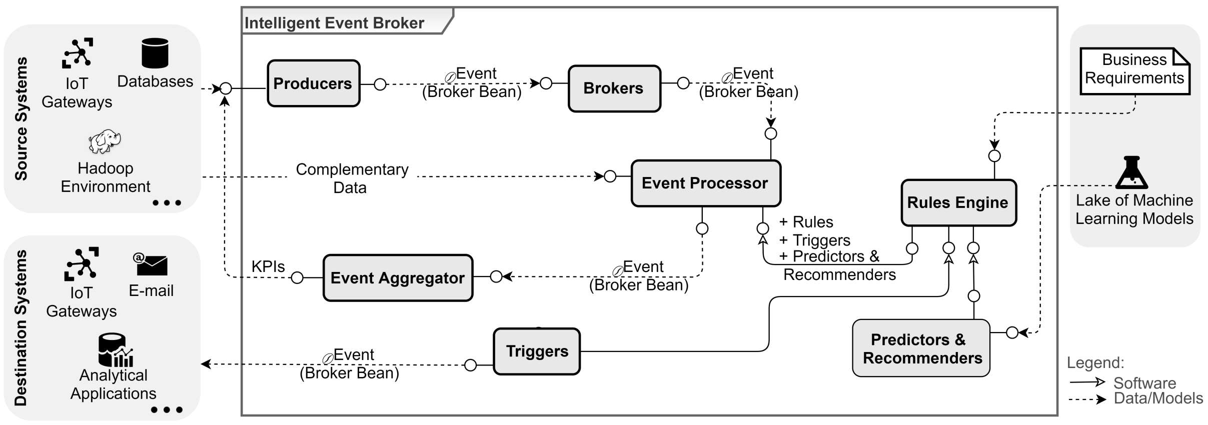


Figure 5-1. Summary of IEB system architecture

At this point, due to the complexity of the *IEB*, Chapter 3 and Andrade et al. (2019) also mentioned a component dedicated to the *IEB* monitoring, perceived to run in parallel to the main components of the system, logging the daily operations of the *IEB*. This system was named *Mapping and Drill-down System* and is composed of a graph database whose code inspector and resulting data model were proposed in Chapter 4 and Andrade et al. (2020) and a Web Visualization Platform that is proposed in Chapter 6 and Rebelo et al. (2019). However, as discussed in the previous section, those works are not detailed enough so that researchers and practitioners can explicitly understand how to implement a logging/monitoring solution for a CEP system in Big Data contexts. For this reason, the work presented here focus exclusively on proposing a logging/monitoring solution so that we can apply it to the *IEB* (or other similar system) and ensure that such CEP system can be adequately monitored and analyzed at scale. The next section details the proposed architecture.

5.3 IEB Logging System Architecture

To create a concise monitoring system, capable of monitoring a complex CEP system like the *IEB*, it is relevant to design an architecture that reflects a set of technical concerns and details (Figure 5-2). Although this work focuses on extending the *IEB*, it must be highlighted that the architecture proposed here aims to structure the main components to achieve an effective monitoring system for CEP systems in the Big Data context, from the collection to the visualization of the data.

Consequently, it is our intention that the community can extract general knowledge from the proposed solution, as the constructs and technologies here proposed can be adapted to work with other CEP systems in Big Data contexts developed in the future. The architecture presented in Figure 5-2, proposes the use of some technologies that were considered the most appropriate for the monitoring system, based on the data that can be extracted from the execution of the *IEB*, which is comprehensibly considered the data source for this monitoring system.

The date and time are the crucial information of the system, representing when something happened in it. To collect the monitoring data, Log4j is typically used in some of the *IEB* components[19] (e.g., the Producers). This technology was selected because it is a widely used (Dickey et al., 2011) open-source and flexible tool, making it an adequate choice for implementing a log system to monitor the *IEB*'s components.

However, some components do not make feasible the use of Log4j to extract the monitoring data, because they are developed using distributed frameworks (e.g., Event Processor developed in Spark), which

[19] More information available in page 3 of (Andrade et al., 2019).

requires a way of centralizing the logs generated by the several nodes in the cluster. In these cases, it is used a Java API.

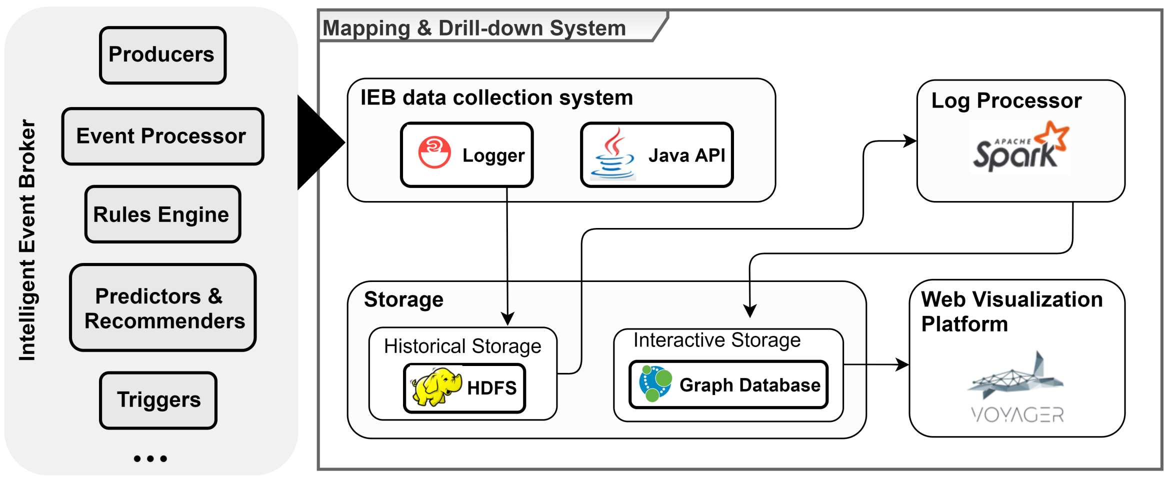


Figure 5-2. IEB logging system architecture

Once the information is collected, the data needs to be stored. One of the most suitable storage systems for the historical and scalable storage of this data is the Hadoop Distributed File System (HDFS). When using Log4j for data collection, Talend Open Studio for Big Data was used as an intermediate tool to send the produced logs to HDFS. When collecting the data through the Java API, it is stored directly in HDFS.

Being the data stored in HDFS, it is relevant to emphasize the use of Spark, where the appropriate transformations are made to the raw logs in a scalable way, as Spark distributes the load throughout a cluster of processing nodes. However, since, at this point, the *IEB*'s data is already collected and stored according to the needs of the monitoring system, there is no need for significant changes to the data contained in the logs. Thus, Spark is mainly responsible for organizing the data to be exported to Neo4j, the technology that supports the graph database of the Mapping & Drill-Down System, since it is the one recognized as the "world leading graph database" (Kumar Kaliyar, 2015).

After performing these steps in Spark, it is possible to visualize a graph in Neo4j with the data related to the *IEB* monitoring. For a clearer visualization, this graph database will feed the Web Visualization Platform developed and proposed in Chapter 6 and Rebelo et al. (2019), named Voyager (with a new interface).

This visualization platform is seen as highly dynamic and immersive, to follow the evolution of the *IEB*, even in contexts of high dimensionality, volume, and complexity.

In terms of infrastructure, a server can be used to execute this monitoring system. To accomplish this, we used Docker as a technology that allows the creation of several containers holding each technology that composes this system. When there is the need for high scalability, all of the proposed components can be deployed in several nodes of a cluster, in order to ensure the needs of a production Big Data context.

This architecture has as a principle the constant and long-term monitoring of a CEP system's operations, and therefore, the collection, storage, and visualization of the data carried out in near real-time. In this way, it is guaranteed that the data is always updated, being possible to draw real and timely conclusions.

5.4 Demonstration Case in the Context of Industry 4.0

Considering the system architecture proposed in section 5.3, a demonstration case was carried out to validate it in the *IEB* context. Previously, Chapter 3 has demonstrated the *IEB* system in the context of Industry 4.0, namely in Bosch Braga, Portugal, hence the demonstration case presented here follows up on this previous one, using Active Lot Release (ALR) data. ALR is a system responsible for supporting quality control during manufacturing and packaging processes in the factory, aiming to classify lots of products as valid or invalid, for them to be adequately delivered to the customer.

To better understand the demonstration case for this work, contextualization of the proposed data models is required. In this context and considering that two storage systems were proposed to store the logs in different phases of the system's monitoring (HDFS and Neo4J), the two data models will be explained below. In addition, some prototype specificities will be discussed, as well as the results that were obtained in this demonstration case. Furthermore, the outcomes achieved with this monitoring system are already properly integrated into the *IEB* Web Visualization Platform previously highlighted in this chapter.

5.4.1 Data Models

HDFS is a file storage system and it fits into the scope of historical storage that is available for analytical purposes when ad-hoc queries are required. Since this work focuses on data generated at high speed, as the *IEB* is a streaming-oriented system, this storage system must respond to this fast production of data, and later, it must ensure fast data search.

The proposed approach for data organization intends to create a table for each *IEB* component, containing the data related to its monitoring. However, as the data collection process was developed in different ways (i.e., Log4j and Java API), the placement of the data in this storage system will also be different. Since the *IEB* Producer component code allows the use of logging mechanisms using Log4j, the data is forwarded to a single log file in HDFS, using Talend Open Studio for Big Data. Each line in the file corresponds to a new event that occurred in the *IEB*, containing the information of the produced event and when it was produced.

Other components make it difficult to use logging mechanisms via Log4j, as they are developed using different frameworks, sometimes distributed (e.g., Spark for the development of *IEB* Consumers and Drools for specification and interpretation of Business Rules in the *IEB*). To overcome this problem, an API was then designed to collect the data, storing it directly on the HDFS. Thus, the monitoring data of a given *IEB* component will go in small files to an HDFS folder (as can be seen in Figure 5-3), reserved to receive the data from the Consumer, and then these small files will be grouped in a single file. However, this HDFS organization strategy may change in the future, if considered relevant, as the monitoring system evolves and becomes more complete. The graph database is used as a storage system for the *IEB* monitoring data and falls into the category of interactive storage. Figure 5-4 shows a representation of the graph-oriented storage system supported by Neo4j, representative of the data collected and transformed for the *IEB* monitoring system, at the time of the *IEB* execution. For a better understanding of the graph depicted in Figure 5-4, it is essential to highlight that the nodes of the graph are represented by circles and the relationships are the links between those circles. Each circle will have a different colour, representative of the *IEB* component (e.g., Producer or Consumer) and the name of the nodes will be the name of the file found in the *IEB* codebase. Time attributes are the key point of this monitoring system since time will be the core property in the relationships between the graph nodes, representing when something happened in the system.

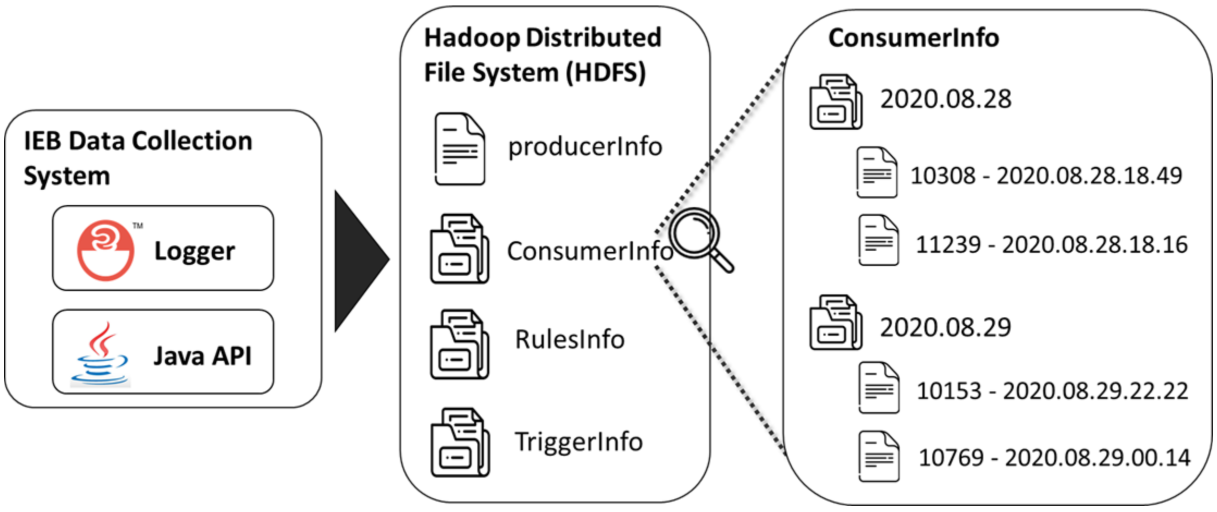


Figure 5-3. HDFS storage system data model for the IEB logging system

In this data model, as can be seen in Figure 5-4 there are three types of Broker Beans (objects representing a piece of *IEB* data, such as a specific event, a business KPI, a business entity, among others): i) ALR Broker Beans that represent ALR events that reached the *IEB* system; ii) Broker Beans representing the calculated ALR KPIs; and iii) Broker Beans created during the verification of the Rules and that are intended to be used later by the Triggers. Thus, the system starts in the *ALRProducer*, the component that produces the ALR events, and, therefore, it is important to store the data containing its behaviour in the system. This component instantiates the Broker Beans that arrive at the system through the ALR events. The same happens in the *AlrOperationalConsumer*, which also instantiates these Broker Beans, and the storage of its behavioural data will also occur, containing the information of the event that has just been consumed.

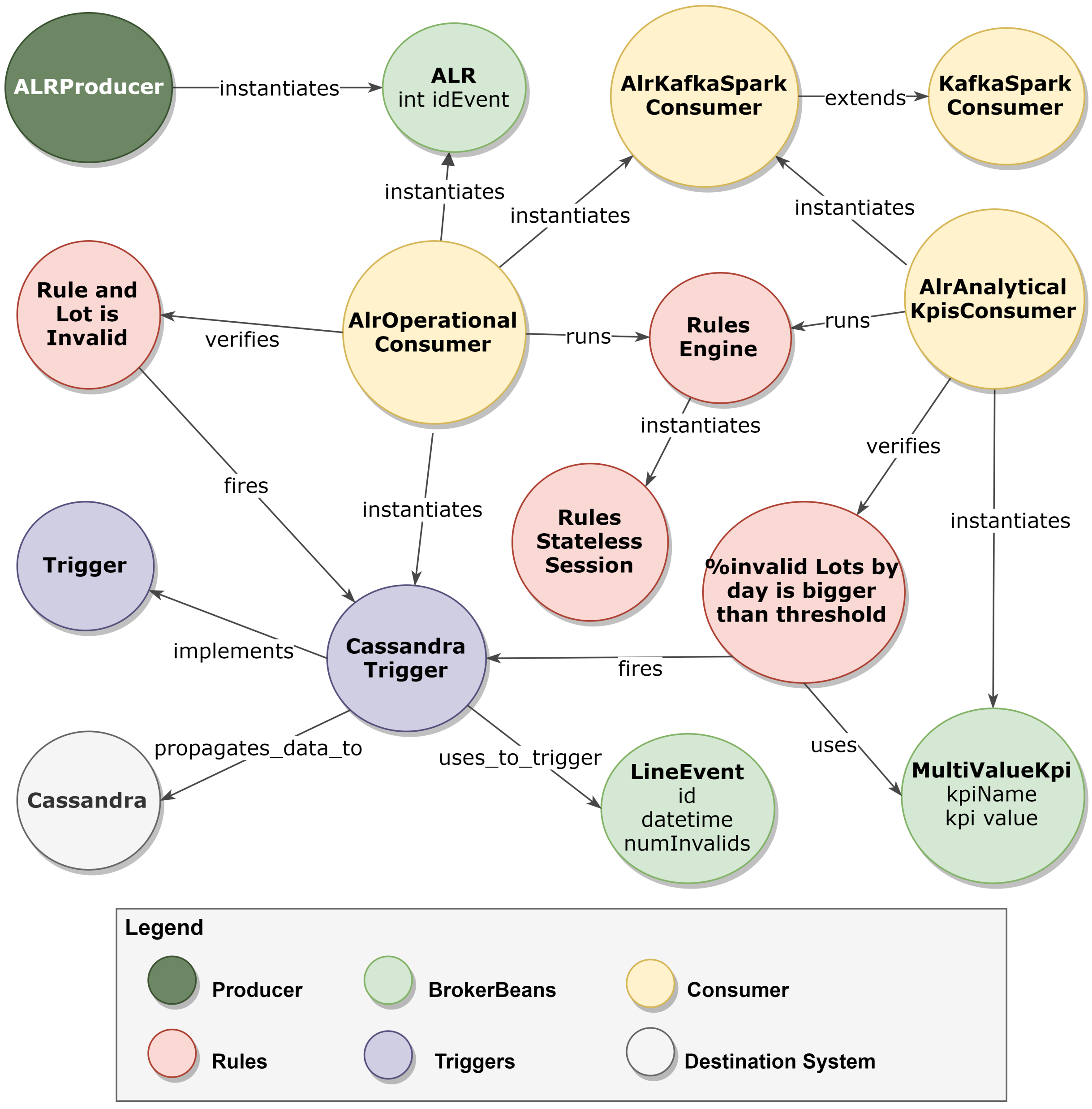


Figure 5-4. Neo4j storage system data model

Operational or analytical Consumers (*AlrOperationalConsumer* and *AlrAnalyticalKpisConsumer*) are constantly waiting for new events and they execute a *RulesEngine* that is responsible for executing Business Rules that have as input the consumed events. The *RulesEngine* class instantiates a *RulesStatelessSession*, which represents a set of all the rules that will be verified by a Consumer. These

Consumers instantiate the Triggers (in this case the *CassandraTrigger* to store data in Cassandra[20]), which can be triggered after the Rules are verified. Regarding the storage of the verified Rules, the name of the Rule and the activated Trigger must be collected, as well as the Broker Bean used for the verification of the Rule, which can be a *MultiValueKpi* Broker Bean or an ALR Broker Bean.

Considering the Triggers, a relationship must be created between the Trigger node (*CassandraTrigger*) and the Broker Bean used to take any action, being the Broker Bean in this case called *LineEvent*. This component implements the Trigger interface and propagates the data to the Destination System (namely the NoSQL database Cassandra).

For further information, the complete data model proposed for the graph can be seen in Chapter 4 and Andrade et al. (2020).

5.4.2 Producers and Consumers Monitoring Process

Regarding monitoring, data collection is the stage that differs the most among the *IEB* components, and it needs proper attention. Considering the Producer's functionality, it is essential to collect some information that makes its monitoring feasible. The Producer's *name* and *id* and the *date* and *time* of the event are essential data for monitoring this component. For this purpose, it is necessary to place the logs in parts of the *IEB* codebase that are considered strategic, extending the *IEB* codebase to collect the information whenever a new event is produced.

After the log data is collected, the same must be stored in a system with the capacity to store large amounts of data, providing high performance in sequential reads and writes, as is the case of HDFS. To store this data in HDFS, Talend Open Studio for Big Data was used, where a job was developed to look for the log file on the server and to store its information in a file in HDFS. This job is executed automatically minute by minute, so that the data that is produced via streaming in the *IEB*, is updated in near real-time in HDFS as well. This way, all the log data generated during the production of an event is stored in HDFS,

[20] https://cassandra.apache.org/

where it will be further transformed via Spark, being later stored in a more interactive and intuitive storage system like Neo4j.

After the data is in Neo4j, it is possible to visualize it in a graph (Figure 5-5), which includes the link between the Producer and the generated ALR Broker Beans. Each relationship indicates the date and time of the Broker Beans production.

To start the collection of the monitoring data in the Consumer (which uses the Spark framework, as previously described), it was considered that the development of a Java API would be an appropriate approach. This Java API allows for the collection of data needed for monitoring the *IEB*, and establishes direct communication with HDFS since when the Consumer component consumes an event, this data is stored on it.

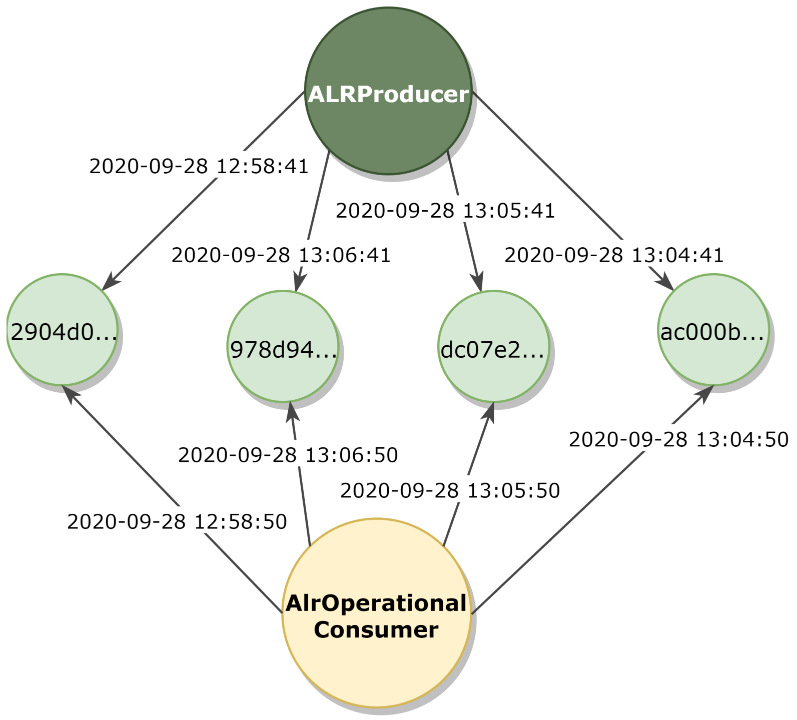


Figure 5-5. Producer and consumer relationships with broker beans

After that, the focus will be the transformations in Spark and the storage of the transformed data in Neo4j. At this point, the graph also contains the link between the events (related to the ALR Broker Beans) and the Consumer, containing the information of the date and time when a certain event was consumed by the Consumer.

After the collection, transformation, and storage of data on Neo4j, both from the Producer and Consumer, it is relevant to analyze how these two systems fit into a single graph. The expectation is that an ALR Broker Bean is produced by the *ALRProducer* and later consumed by the *AlrOperationalConsumer*, resulting in an event with two connections: one related to the production time and the other one related to the consumption time. The graph produced by the two components is represented in Figure 5-5 where it is concluded that the result was the one expected. This data is in constant growth since it is collected and stored in a streaming context.

5.4.3 Data Representation in the IEB Web Visualization Platform

Since the data is adequately stored in Neo4j, this graph is the data source for the Web Visualization Platform proposed in Chapter 6 and Rebelo et al. (2019). When accessing the platform, a connection to the Neo4j database and validation of the data can be made, and after that, the user can freely explore the graph.

The first exploration on the platform presents a statistical summary of the data presented in the accessed database. In this exploration, information about the attributes of the Neo4j database is presented, both information about the nodes and information about the existing relations between them. In this same component, the user can develop queries to collect the needed information regarding the monitored data.

As can be seen in Figure 5-6, in the "Visual Query Viewer" component, it is possible to visualize, in a generic way, the quantity of the data available in the graph (on the left). In this specific case, 1 Producer and 1 Consumer were logged, as well as 22.047 events. In the centre of the screen, the organization of that data is available by an interactive graph that evolves as the nodes are clicked, providing a more immediate interpretation when navigating through the data.

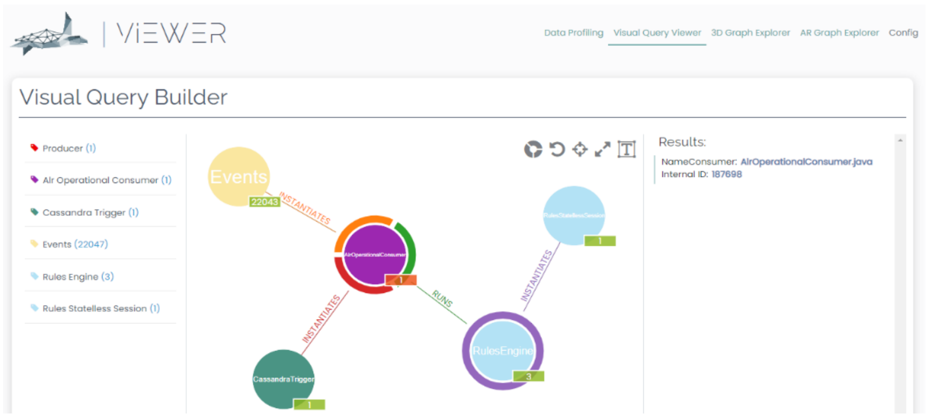


Figure 5-6. Voyager visual query viewer presenting the monitoring data

The *AlrOperationalConsumer* focused on the centre of the graph (the purple circle rounded by red, green, and orange) instantiates 22.043 events. Only with this information can be concluded that, if exists 22.047 events in the database and only 22.043 were consumed there are 4 events that, at this point, already were produced by the *Producer* but were not yet consumed by the *AlrOperationalConsumer*.

Besides the representations mentioned so far, there are more components developed in the platform which allows visualizing the graph developed in Neo4j with other techniques. For example, Figure 5-7 show us all the ALR Broker Beans (yellow nodes), the *ALRProducer* (red node) and the *AlrOperationalConsumer* (purple node) using the 3D concept. Other techniques such as Augmented Reality are used in other components of the Web Visualization Platform (Rebelo et al., 2019).

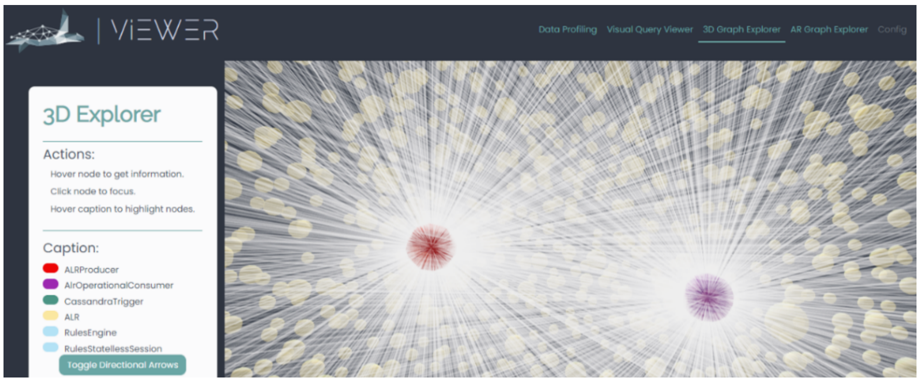


Figure 5-7. Voyager 3D graph explorer

As seen, the visualization component of this work is focused on several graph visualizations demonstrated so far, including significantly enriching analyses on the data that flows throughout the *IEB* components, hence making it possible to monitor what is happening in a CEP system in Big Data contexts.

5.4.4 Demonstration Case Insights and Discussion

The demonstration case explored in this section is used as proof-of-concept for the CEP logging/monitoring architecture proposed in this work, targeting Big Data environments. Through the exploration and analysis of the usefulness and efficacy of this demonstration case, some insights are relevant and, therefore, discussed here in this subsection.

In general, and considering the analysis of the insights retrieved from the functioning of the system throughout this demonstration case, we can conclude that the proposed architecture can be considered significantly adequate to ensure its main purpose, i.e., adequately log and monitor the daily operations of a CEP system in Big Data contexts. The demonstration case has shown that the system could adequately collect and process a vast amount of log data without a significant increase in the processing time or even a decrease in the overall performance of the system.

Although there are no signs of an increase in the processing time of the step that collects and stores data in HDFS, once we increase the amount of data, the same doesn't happen in regard to the Spark processing and Neo4J storage. In this specific step of the process, the processing time increases as the amount of data also increases. In the created scenario, when we have a small amount of data in HDFS, the Spark processing and Neo4J storage are made in fractions of seconds but, when we increased the amount of data 5000 times, we have noticed an increase of 13 seconds in the processing time. However, since we are using Big Data technologies in our architecture, to achieve lower processing times, we can simply scale-out our Spark and Neo4j clusters, reason why combining CEP concepts with Big Data technologies and paradigms is so relevant.

Regarding the visualization platform, and when comparing it directly with the Neo4J option for the visualization of the graph database, we can conclude that their behavior is similar, presenting the data for analysis almost immediately, without any noticeable delays. When considering the visualization of the whole graph and when it exists a huge amount of data, the visualization platform needs more time to organize the graph, which is understandable. The increase we have registered in those cases was around 10 additional seconds, highlighting again that in such scenario we had 5000 times more data, which may rule out the occurrence of a performance issue, as several filtering techniques can be used in the visualization platform to avoid the creation of such massive graphs.

5.5 Chapter Summary

Taking into account the work developed so far, and the insights gained during the development of this monitoring system and its demonstration via prototype in the context of Industry 4.0, the proposed architecture can be considered as suitable for solving the identified problem, i.e., the need to ensure adequate monitoring of CEP systems in Big Data contexts.

Given the complexity of the *IEB* in production contexts, it is considered essential to ensure adequate monitoring of the system. This monitoring was supported by storage, processing and analytical technologies and it is part of the proposal for the *IEB Mapping and Drill-down System* depicted here. In this way, it is now possible to see what is happening in the *IEB*, or, when generalized, in any CEP system for Big Data contexts, making it easier to control the system and to visualize the data flowing throughout the components and at which scale.

It is therefore considered that this work may be relevant for future studies that focus on bridging the gap between CEP and Big Data, making monitoring of CEP systems in Big Data contexts more accessible to the community.

Chapter 6. The Immersive Web Visualization Platform for the Intelligent Event Broker

This chapter[21] presents the work related to the design of the Web Visualization Platform that provides the analytical and drill-down capabilities on the Monitoring System of the *IEB* (6th objective of this thesis, presented in Chapter 2 and in Andrade (2019)). In this chapter, it is presented the architecture for the Web Visualization Platform, the description of its several components, and a demonstration case in the context of Bosch Car Multimedia.

6.1 Introduction

As the Industry 4.0 movement emerges and popularizes, organizations are trying to make use of the data produced inside and outside of their facilities to improve their business processes and, consequently, to achieve better results and increase their business value. Considering this movement and the related technological evolution, machines or other devices used in organizations are already producing data in real-time and enabling its processing immediately after the occurrence of various events. With stream processing, data produced by Internet of Things (IoT) devices will be useful for organizations by delivering value in a short time frame and by providing insights about the status of the organization's processes and environment.

However, having data is not synonymous of organizational improvement. In fact, having several data sources with different formats or data being produced at different velocities can bring chaos to the organization. To avoid this chaos, a system that enables event processing in real-time is crucial to improve the organizations' daily operations, considering that, nowadays, data variety, volume, and velocity are key

[21] This chapter integrates the content of the research paper "An Immersive Web Visualization Platform for a Big Data Context in Bosch's Industry 4.0 Movement", published at The European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS'2019) (Rebelo, Andrade, Costa, & Santos, 2019). The contributions of Carina Andrade for this work were focused on the global definition of the architecture for the Web Visualization Platform and implementing the graph database making the data available for the implementation of the demonstration case. **Acknowledgements.** This work has been supported by national funds through FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2019 and the Doctoral scholarship PD/BDE/135101/2017 and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 039479; Funding Reference: POCI-01-0247-FEDER-039479]. Authors acknowledge the Virtual and Augmented Reality work of Ana Dias, André Domingues, Maria Cardoso, Rui Faria and Vanessa Ferreira, the Bosch Car Multimedia and the 8thWall team that supported our work during the development phase.

aspects of event processing. In this context, an Intelligent Event Broker (*IEB*) system was proposed in Chapter 3 and Andrade et al. (2019), serving as a guide for the implementation of Complex Event Processing (CEP) systems in Big Data contexts. In the logical and technological architecture of the system, besides the components related to the data collection, processing, and aggregation, it was considered a Mapping and Drill-down component for monitoring the evolution of the *IEB*. With this component, the chaos that can result from deploying such systems at scale can be minimized using innovative and immersive visualization components, both generically and idealized for Industry 4.0 scenarios. This work is focused on the proposal, demonstration and validation of the immersive visualization system.

The relevance of this type of visualization system in the Big Data era and its proposed system architecture are explained further in this chapter. The roles of the system components are also explained, as well as their function within key areas of the *IEB* deployment context. The proposed visualization system is followed by a demonstration case in the production lines of Bosch Car Multimedia Portugal, an organization currently aligned with the Industry 4.0 movement.

This document is structured as follows. Section 6.2 presents a brief introduction to the *IEB* and the related challenge that motivates this work, as well as an identification of challenges regarding data visualization in Big Data and their relevance to the challenge of this chapter. Section 6.3 presents the architecture for the Web Visualization Platform proposed in this work. Section 6.4 presents the demonstration case at Bosch Car Multimedia Portugal. Finally, Section 6.5 summarizes the presented work.

6.2 Conceptual Background

6.2.1 The Intelligent Event Broker

The *IEB* proposal consists of a CEP system specially designed for Big Data contexts, having the goal of providing an event processing-oriented architecture to the age of high data volume, variety and velocity. Its system architecture combines a typical CEP-based architecture with flexible and scalable Big Data technologies and techniques (Andrade et al., 2019). Such a typical CEP architecture comprises three principal areas: a collection of *source systems*, an intermediate processing unit and a collection of *destination systems*. In the *IEB*, *Source Systems* can be any relevant physical devices or virtual data sources such as IoT Gateways, APIs, databases, Hive tables or HDFS files. Similarly, *Destinations Systems*, can also be relevant physical devices, data stores or relevant business applications, such as E-Mail, SMS or analytical dashboards.

The broker component itself is *IEB*'s area dedicated to event collection, event processing, event aggregation, rules verification, application of machine learning models, and execution of triggers. i) The *Producers* component is designed to standardize the collection of events using Kafka (kafka.apache.org/), a distributed streaming platform. Here, events are serialized into the form of broker beans representing the business entities. Then, they follow their path to be published in a Kafka topic that is stored in a cluster of Kafka brokers. ii) The *event processor* (Kafka consumers embedded into Spark applications) subscribes Kafka topics and is always listening for new events, regardless of frequency. iii) This component is also responsible for the runtime translation of *rules* that are defined in Drools (drools.org), a *rules engine* technology. These rules reflect the business requirements and can be classified as *strategical*, *tactical*, and *operational*. iv) Complementary data arriving from any *source system* can be used to bring additional insights to the business requirements that are being evaluated. v) The *triggers* component is related to the *rules engine*, and it is responsible for the connection to the *destination systems*, performing actions due to the application of the rules. vi) The *event aggregator* component is responsible for the data aggregation. Druid (druid.io) is the suggested technology to perform the aggregations and to store them for the calculation of *Key Performance Indicators* (KPIs), since it is a columnar storage system already explored and considered as being able to aggregate huge amounts of event data with sub-second response times (Pyne, Rao, & Rao, 2016; Tsai, Lai, Chao, & Vasilakos, 2015). vii) The *predictors & recommenders* component is considered as being relevant in this type of system, ensuring the possibility of introducing *machine learning* capabilities into event processing systems. This component considers the existence of a *lake of machine learning models* that were previously trained and that are applied to events according to what is defined in the rules.

Considering the expected constant increase in data and complexity resulting from the system behavior throughout its daily activities when deployed in contexts like Industry 4.0, smart cities, agriculture, among many others, the *IEB* system architecture additionally proposes the introduction of a *Mapping & Drill-down System*, parallel to the main broker component. This system's goal is to allow the constant and long-term monitoring of the *IEB*, even in greatly complex scenarios. To achieve this component, the *IEB* proposes the implementation of a *Graph Database* whose data model is based on meta-data from the *IEB* with the mapping of all its existing components, their relationships, and instantiations. This will serve as the data source for a *Web Visualization Platform*, which constitutes the challenge of this chapter.

As the architecture of the *IEB* was already instantiated with a demonstration case at Bosch Car Multimedia Portugal, using event data from its Active Lot Release system, this chapter's demonstration case will expand the previous one, using the same data.

6.2.2 Data Visualization in Big Data and its Relevance to the IEB

Although there are still many challenges related to the Big Data characteristics, the value of analyzing Big Data is widely recognized in the scientific community (Pyne et al., 2016). The main differences between Big Data Analytics and traditional statistical analytics are related to the data volume, its high dimensionality, complexity, and data variety, as well as the noise, errors, or other problems found in the data. However, more data is not synonymous of better information or better decisions, being necessary to find new ways for visualizing Big Data and face the identified challenges (Tsai et al., 2015).

Regarding the challenges directly related with visualization in Big Data contexts, Gorodov & Gubarev (2013) mention: i) the visual noise created by the data volume available for analysis; ii) limitations of the presentation devices (e.g., screens dimensions) and of the user's physical perception; iii) the information loss due to aggregations performed for dealing with the data volume; iv) increased hardware needs to compute dynamic visualizations; and, v) high rate of changes when the visualizations are connected to dynamic data sources. The work of Agrawal, Kadadi, Dai, & Andres (2015) agrees with those challenges, also associating them with the needed scalability to migrate from the traditional visualizations to the visualizations in the Big Data era: i) scalability of the perception (limitations related to the user's perception and to the visualization devices); ii) real-time scalability (challenges related to the visualizations' changes due to the vast amount of data that arrives at the system in real-time); and, iii) scalability of the interaction (limitations related to the interaction response time due to the high latency of the visualization mechanisms).

Some strategies to deal with these challenges come from novel Big Data systems that are designed with such issues in mind (like the *IEB*); from new conceptual strategies to visualization; and from innovative technologies and devices. Conceptually, both previously mentioned authors refer to choosing the right type of representations for the data, sampling, filtering or parallelization as examples of such strategies. The most innovative trend of research, both conceptual and technological, is Immersive Analytics. This topic is dedicated to the exploration of new interaction and presentation technologies for decision support (Chandler et al., 2015) as well as the study of the potential of virtual and augmented reality technologies, the use of large touchscreen displays, or even the use of multisensory infrastructures offering a more immersive experience to users (Chandler et al., 2015; Sicat et al., 2019). Some examples for the practical use of these techniques may be: i) the use of immersive technologies for the collaborative analysis of multidimensional data (Butscher, Hubenschmid, Müller, Fuchs, & Reiterer, 2018); ii) the use of virtual reality technologies and platforms for scientific data visualization (Donalek et al., 2014); and, iii) the creation of *In situ* visualizations using augmented reality (ElSayed, Thomas, Marriott, Piantadosi, & Smith, 2016).

Another relevant consideration is the growing trend of using Web development for the creation of data visualizations (Mei, Ma, Wei, & Chen, 2018). This new trend takes advantage of the evolution of Web programming languages to implement dynamic, responsive, and interactive data visualizations (Shahzad, Sheltami, Shakshuki, & Shaikh, 2016). Solutions such as D3.js (d3js.org/), ECharts (echartsjs.com), or the Data-Driven Guides (Kim et al., 2017) are examples of different Web-based approaches for the current challenges of Big Data visualization as well as the adoption of native Mixed Reality capabilities and performance improvements on modern browsers that brought the Immersive Analytics concept to the Web (Butcher & Ritsos, 2017).

In this context, and as already highlighted in the introduction, the need for a system like the *IEB* arises from the technological evolution and the Industry 4.0 movement, which are transforming the factories far beyond what we once knew. However, it is required that such a system (and its data) can be easily and (semi)-autonomously managed and monitored. This vision for the management and monitoring of a unified system using innovative and immersive visualization techniques was not yet identified in any related work. In fact, the only work that mentions the use of dashboards in a system that aims to integrate CEP and Big Data concepts was the work of Ioannis Flouris et al. (2016). However, in this specific case, the dashboards are not being used for the management and monitoring of such system, nor do they have the level of innovation and functionality presented here. For this reason, to the best of our knowledge, this work can be significantly useful for advancing the state of the art regarding data visualization for the (meta) management and monitoring of CEP systems in the Big Data era.

6.3 Architecture of the Web Visualization Platform

Motivated by the previously mentioned challenges that affect the long-term, real-time, continuous monitoring and management of an *IEB* deployed in production environments, a Web Visualization Platform based on novel and immersive technologies and techniques is instantiated as a Web-based system architecture, proposed to fulfil the previously described role of *Mapping and Drill-down System* of the *IEB*. The presented approach for this *IEB* component considers a previously defined and developed Graph Database (out of the scope of this chapter) and the here proposed Web Visualization Platform composed by an API layer and a Web Application layer.

The API layer leverages the connection between the Web application and the graph database. Due to the high variety of technological options, the development of this component should be based on the use-case, available human and business resources and complexity of the infrastructure. Options for the development of this layer can vary between the simple usage of native database REST APIs or prebuilt drivers, like the Neo4j Bolt Driver for JavaScript (git.io/JeoJ5), directly in the application code for easier direct queries (at the expense of harder maintainability); or the development of more robust solutions based on a custom REST or GraphQL (graphql.org) API, that would allow for separation of functions and roles between layers, abstracting the data model and structure from the frontend and even allowing the integration of future additional data sources, like monitoring data from specific component software whose data might be relevant but not applicable or suitable to the graph database's data model.

The Web Application layer described below is divided in four fundamental areas of visualization, where relevant conceptual visualization components are proposed. Further demonstration and validation of those components are presented in section 6.4. The four main areas are presented in the following subsections. Figure 6-1 presents the conceptual architecture of the Web Visualization Platform with the previously described layers and proposed components within the context of the *Mapping & Drill-down System*.

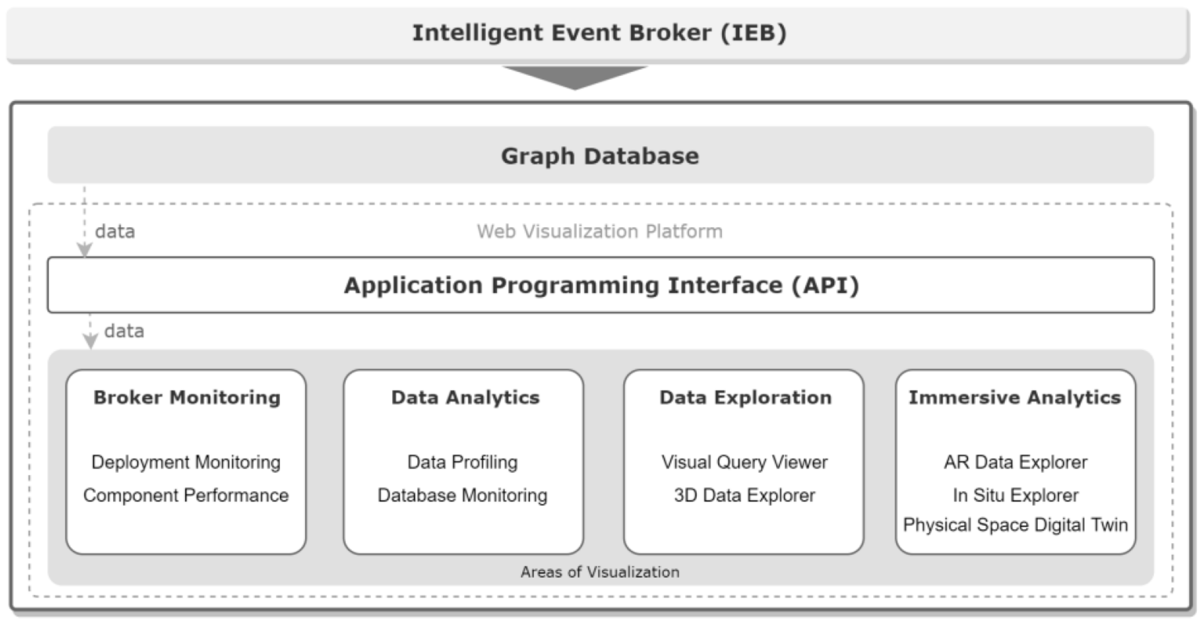


Figure 6-1. Conceptual architecture of the proposed Web Visualization Platform

6.3.1 Broker Monitoring

The goal of components in this area is to provide continuous monitoring of the *IEB* infrastructure in production-ready environments. As detailed in Section 6.2, a typical *IEB* deployment includes several individual *software* solutions that often come with built-in monitoring metrics and mechanisms. Similarly, deploying the *IEB* in production will demand a local or remote hardware infrastructure whose monitoring can be critical, especially as the complexity of the *IEB* grows over time. Two components are here proposed:

- The Deployment Monitoring is a component to monitor the deployment infrastructure that houses an *IEB* instance, particularly detailing relevant information regarding resource usage of both hardware and software. Development of this component should be based on logs, metrics and rules gathered from sources such as system metrics (e.g. CPU usage, memory use, disk space usage, etc.) or, in cases of remote/cloud-based deployment, through connections to built-in or personalized monitoring and telemetry services, as available in solutions like Amazon CloudWatch or Azure Monitor;

- The Component Performance is a component dedicated to monitoring the internal performance of the software from broker components (e.g. Apache Spark, Apache Kafka, etc.). Such an analysis guarantees detailed monitoring of both broker and individual component performance, allowing for possible detection of system bottlenecks or business-related anomalies. Development of this component can be achieved via custom metrics, addition checkpoints in the broker architecture codebase and/or through native metrics and tools preconfigured in the component's software.

6.3.2 Data Analytics

Components in this area are related to the graph database and its data structure. Interactive graphical representations and editable temporal overviews from data gathered and stored granularly over time should allow long-term monitoring of both the data model and database technology. Conceptually, two different components are proposed:

- The Data Profiling component aims to provide a holistic view over data based on statistical analysis and informational summaries of relevant parts of the dataset. This should allow for tasks such as continued assessing of data quality, identification of patterns, and pure metadata analysis.

- The Database Monitoring component is proposed to make use of metrics, logging, and other analytics mechanisms provided by the chosen database technology to continuously monitor its performance in matters such as of transaction information, storage usage, or query performance.

6.3.3 Data Exploration

Exploration components are meant to offer new and customized approaches to traditional visualization strategies in desktop and touch-enabled devices. By leveraging the naturally understandable visual properties of graphs and its relationships between nodes, two conceptual components are proposed:

- The Visual Query Viewer as a component intends to provide an interactive, query-based, visual approach to graph exploration, bridging the gap for non-technical users not proficient in the relevant query language(s) by allowing the dynamic visual selection of nodes and relations that, on the background, translate the actions into proper language-specific queries and return the proper results in a familiar, easily interpretable fashion.

- The 3D Data Explorer component devises a tridimensional representation of the graph structure that represents the data, using volume and a third interactable axis to provide a step further in data exploration. Even using traditional interaction techniques in desktop and mobile devices, such a component allows a more immersive, interactive, and deeper analysis than the possibilities offered by simple 2-dimensional non-volumetric representations of a graph.

6.3.4 Immersive Analytics

These innovative components provide novel approaches for visualization, presentation, and interaction, by leveraging mixed-reality environments and techniques, resulting in an enhanced solution to the challenges provided by the related context. Three innovative components are here proposed:

- An AR Data Explorer component aims to bring the 3D Data Explorer component one step further by allowing the exploration of the graph database in an augmented reality environment which results in an even deeper immersive exploration environment and its application in newer visualization formats (like a Head-Mounted Display) and interaction techniques (like voice and gesture-recognition).

- The *In Situ* Explorer, given the industry-bound theme of the project, is proposed as a way to present virtually generated information on top of location-specific physical components, providing an augmented reality experience that offers live and on-site relevant information, alerts, or status changes to the user.

- A Physical Space Digital Twin is proposed as a Virtual Reality enabled facsimile of the factory plant that allows the user to remotely explore it as if he/she is present on-site, providing a real-time link to production performance and factory component monitoring.

Flexible, extensible, and diverse solutions from the vast ecosystem that is modern Web development allow for a wide range of different options when implementing a practical demonstration of the proposed architecture and its components, both in the open source and propriety software domains. The goal of this proposal is to leverage both mature and innovative software solutions for the Web to deliver next-generation visualization experiences for the end-user, as it will be presented in the following demonstration case. However, within the proposed architecture, the use of different software stacks and technological components is not only possible but highly incentivized, as long as those efforts better suit the use case and the organization.

6.4 Demonstration Case at Bosch Car Multimedia Portugal

A demonstration case for the proposed system was developed at Bosch Car Multimedia Portugal by adding a fully functional *Mapping and Drill-down* component to the work previously started and presented in Chapter 3 and Andrade et al. (2019). Connecting to the previously existing Neo4j database, arises the development of an Express.JS (expressjs.com) multi-page Web application, serving EJS-based views on the client-side, supported by custom JavaScript logic and imported libraries/frameworks. Relevant general features of the Web application include: i) encrypted and secure HTTPS and BOLT (Neo4j's own communication protocol) connections by implementing SSL certificates; and ii) platform/environment abstraction by containerizing the application using Docker, allowing for ease of development, testing, and deployment in a secure, remote, and cloud-based infrastructure.

To demonstrate the design simplicity, the flexibility of the development, and the variety of development options proposed in the architecture, the demonstration case is mostly based on open-source libraries, frameworks, and solutions. The Web application features a design language that follows Bosch's brand guidelines achieved through the development of a custom Bootstrap 4 (getbootstrap.com) theme customized using Sass. This demonstration case focused mainly on the most interactive and immersive features of the proposed architecture, having the following components been the most extensible developed and tested.

6.4.1 Visual Query Viewer

This component is based on a design and implementation modification of PopotoJS (popotojs.com), an open source JavaScript library built over D3js. The library connects to the Neo4j database via secure REST API connection and automatically manages the interaction and communication between the database and the application.

The *Visual Query Viewer* component features three main interactive sections: Graph, Query, and Results. The *graph* is the main section that presents the graph database's taxonomy (based on Neo4j labels) and an SVG-based interactive navigation of the graph that allows technical and non-technical users to query the database using actions like selecting/deselecting a value, adding/removing relationships between nodes, and using logic gates. The *query* section presents a text-based approach to the user queries, both using a natural language approach (starting with "I'm looking for...") but also showing the correspondent Cypher query (Neo4j's query language) that composes the current visualization. Lastly, the *Results* section presents a custom-designed view of the query results matching the current state of the interactive graph query. In the *IEB* context, the *Visual Query Viewer* component allows the exploration of the data available in the graph database, which reflects the data model of the *IEB* itself. Using this component, the instantiations of the several *IEB* components can be explored in an easy and intuitive way. In Figure 6-2 this component can be seen in action within the general application layout and design. As an example, the figure shows the relationships around the instantiation of "ALR Producer" as a *Producer* component of the *IEB*. Related *broker bean* and *event producer* component instantiations are shown as the user progressively drills-down the information, tracking the cycle of an event through the system.

6.4.2 3D Data Explorer

The *3D Graph Explorer* component leverages an open-source 3D Force-Directed Graph Web component (git.io/fjywM) allowing for the tridimensional rendering of a graph data structure using an interactive force-directed layout. Using the official Neo4j JavaScript driver, the 3D Graph Explorer implements a customized implementation of the 3D Force-Directed Graph library backed by data retrieved from the Neo4j database. Other features of this component include real-time WIMP (Windows, Icons, Menus, and Pointer) and post-WIMP based interaction with the graph and live structure updates, including the addition of labels representing the relationship between nodes, arrows to indicate the directionality of these relationships, and label-based color-coded nodes.

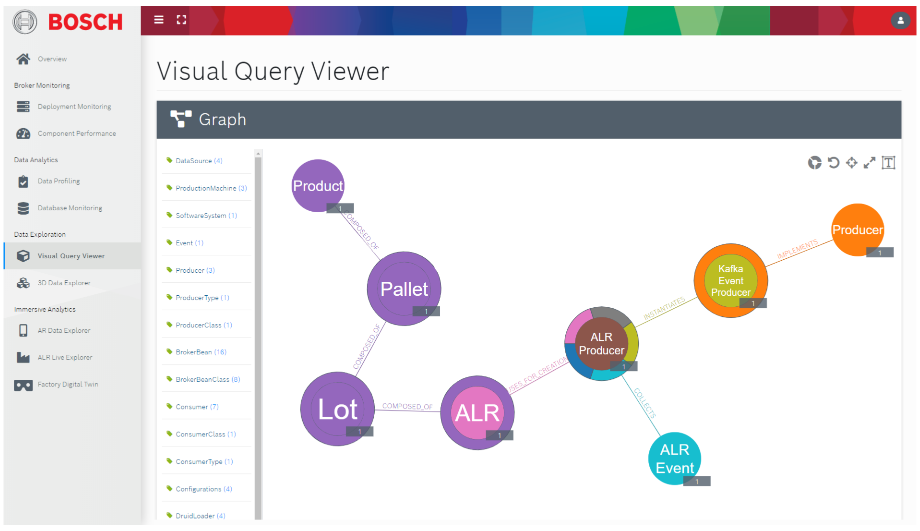


Figure 6-2. Visual query viewer component using Bosch's layout

6.4.3 AR Data Explorer

Following the approach from the 3D Graph Explorer, this component leverages a fork of the same 3D Force-Directed Graph Web component specifically tuned to work as an A-Frame component (git.io/JeoJy). A-Frame (aframe.io) is an open-source Web framework for building Virtual Reality experiences on the Web by registering components with a declarative and extensible structure using HTML-like semantics. Having the graph visualization within an immersive and positionally-tracked environment, the *AR Data Explorer* component becomes an Augmented Reality experience by introducing 8th Wall Web (8thwall.com/web), an Augmented Reality platform for the mobile Web that uses only Web standards to implement a proprietary Simultaneous Localization and Mapping (SLAM) engine, 6-Degrees of Freedom (6DoF) Tracking, Surface Estimation, Image Tracking, and other features for the real-time AR on the Web. 8th Wall Web, being the only proprietary software in the demonstration case, can expand and integrate multiple 3D JavaScript frameworks, including A-Frame.

Other open-source solutions, namely ARjs (git.io/Je4Vp) were tested and considered. However, although capable, ARjs is a limited library that works only for marker-based applications and does not feature the advanced tracking and positional capabilities of 8th Wall Web that allow for markerless experiences. Extra features and conveniences like an extensible API, built-in event handling methods, high quality documentation, and multiple easily available channels for help and support are also advantages of 8th Wall Web.

For the demonstration case, the AR Data Explorer (Figure 6-3) is comprised of an 8th Wall Web markerless/world-tracking SLAM based scene that superimposes the 3D force-graph over the tracked real-world captured by the users' camera-device. The custom user interface allows for the repositioning of the graph in the scene, allowing for adjustments based on the user's environment and supports touch-enabled interactions for selection, resizing, and rotation as real-time interaction in an AR environment. Furthermore, considering the ubiquity of the Web in Internet-enabled devices, the demonstration case features a custom post-WIMP based method of voice interaction.

Inspired by popular current digital assistants, the component features a demonstration of the implementation of the open-source library Artyom.js (sdkcarlos.github.io/sites/artyom.html), a convenient wrapper of the recent *speechSynthesis* and *WebkitSpeechRecognition* APIs that allowed for the introduction of hands-free interaction by using voice detection and voice commands to control and alter the augmented reality scene. Voice-based interaction, especially continuous mode-based solutions (which Artyom.js supports), allows for the usage of the system in Head-Mounted Displays that may not feature (or require) hand-based input controls.

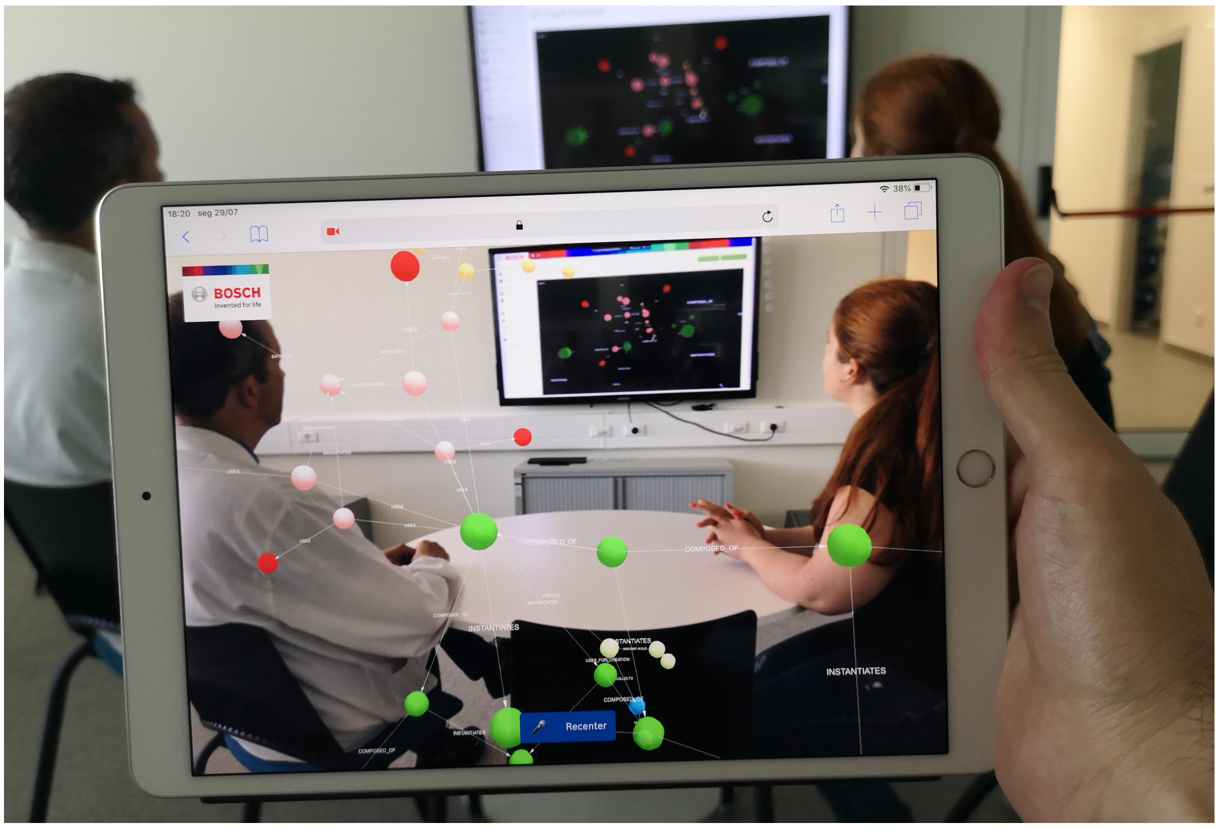


Figure 6-3. 3D graph explorer and AR data explorer demoed at a Bosch meeting

The AR Data Explorer, demonstrated in Figure 6-3, allows for a deeper, closer, and more immersive interaction with the graph database, supporting even a large-scale expansion. In this context a user can figuratively insert himself inside the graph and interactively explore its information as it surrounds him. This constitutes an intuitive and interesting way of data exploration, for example in teamwork environments such as brainstorming sessions, problem-solving discussions or sprint planning meetings.

6.4.4 In situ Explorer

By considering the layout of Bosch's newest factory at Bosch Car Multimedia Portugal in Braga and its production lines, the *In situ Explorer* component (Figure 6-4) features an innovative approach to Augmented Reality on the Web, by leveraging both the markerless capabilities and the recently developed Image Target (marker based) capabilities of 8th Wall Web for a novel approach to a large scale tracking solution on an industrial setting. Image-based markers are used for starting the application and setting the virtual world in the correct positions over the real-world setting, while SLAM-based smart recentering allows for the user to move over greater distances within the scene while avoiding significant drift of the virtual environment.

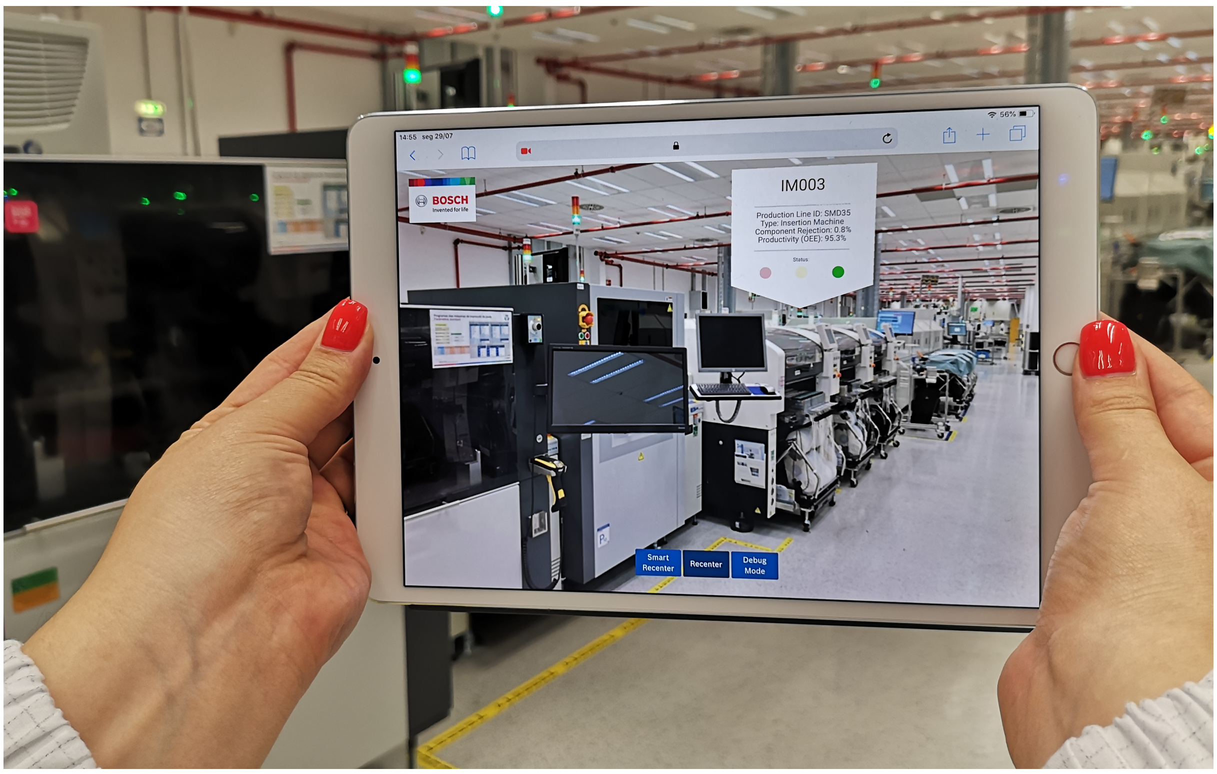


Figure 6-4. ALR live explorer component at the Bosch Car Multimedia Portugal facilities

As demonstrated in Figure 6-4, the *In situ Explorer* tracks the position of factory machines and components, and presents relevant information on-site overlaid on top of the correspondent component, providing immersive and attention-grabbing access to real-time, animated, and easy to understand critical information on the factory floor to the relevant users. In this specific case, we can see a Bosch production line presenting different types of machines (e.g., inspection machines and insertion machines). In one of the insertion machines, a label with relevant information can be seen above it, showing the component rejection and productivity percentage at that moment. For a better understanding of the presented data, a green ball is shown representing a positive situation for the overall machine status. With this type of interaction with the production lines, the production control can be done in an easy way, with just-in-time actions.

This demonstration case has allowed to test how the system would function on a real use-case scenario. As intended, the developed interfaces allowed for an interactive, immersive exploration of the mapped *IEB* components and instantiations. The also allowed the leveraging of component representations as an intuitive graph for easy drill-down of data. Additionally, as shown in the figures, the developed case was presented at Bosch Car Multimedia Portugal facilities and shown to relevant professional and stakeholders, both from the factory and from the *IEB* development team, from which positive feedback was received on this proof-of-concept.

6.5 Chapter Summary

This chapter shares the proposal for the design and implementation of a Web Visualization Platform for a Big Data/CEP system, motivated by current needs in the Industry 4.0 movement and challenges in the Big Data visualization realm. Since the system proposed in this work is being developed as a part of an *IEB*, a contextualization of the *IEB* architecture was carried out to share the motivation for the system and its design details. The exploration of the challenges of data visualization in Big Data contexts, as well as the directions found in the literature review to surpass them were analyzed. As a contribution to the state of the art, this chapter presents and fully explains a proposed architecture for meta-monitorization and management of the *IEB*, by dividing it in key areas where innovative, immersive techniques are leveraged to create next-generation visualizations on the web as conceptual components. Moreover, a demonstration case at Bosch Car Multimedia Portugal is presented to validate the proposed architecture and to present an example of a possible technological and logical approach to the development of the entire *Mapping & Drill-down System* component of the *IEB*.

Chapter 7. Conclusions

This chapter focuses on presenting a summary about the undertaken work and the achieved results for the specific problem of Bosch Car Multimedia. Furthermore, this chapter summarizes the future work and lists the publications representing the papers with the scientific contributions and other studies of this thesis, reviewed and accepted by peers of the scientific community in the Information Systems field of study.

7.1 Undertaken Work and Achieved Results

As previously mentioned, this doctoral thesis integrates several chapters that address different research papers that were reviewed and accepted by the scientific community. Throughout these chapters, the objectives of this thesis were completely or partially achieved, being this section a discussion and conclusion about the undertaken work and the achieved results in each of those works.

The work depicted in Chapter 2 demonstrated the need for the exploration of the Complex Event Processing (CEP) topic in Big Data contexts. To accomplish that, some works identified in the state of the art were detailed, and it was highlighted the main contributions of this research work for the scientific community. This research paper (Andrade, 2019) served as the basis for the following contributions as it sets the main goal and objectives of this doctoral thesis as well as the research methodology that was used.

Chapter 3 presented the architecture of the Intelligent Event Broker (*IEB*), our proposal for a CEP System in Big Data Contexts. The conceptual and technological components were specified and their role in the architecture was explained. At this point, a demonstration case was carried out in the context of Bosch Car Multimedia, to assess the suitability of the defined architecture. The data from the packaging and quality control process (Active Lot Release system) at Bosch was used as an example for the definition of the rules that were integrated in the demonstration case. With this work and demonstration case, a first version of the system was implemented, having fulfilled the 1st, 2nd, 3rd, 4th and 5th objectives presented in Chapter 2. The *Producers* component, considered in the presented architecture (Figure 3-1), allows for the integration of several data sources into the system and, therefore, accomplishes the 1st objective, since it can be defined a producer for each type of communication interface facilitating the integration of different data sources. Regarding the consideration for data volume, variety and velocity, although no benchmark was carried out, the system architecture proposes the usage of Big Data technologies (e.g., Spark and Kafka) that were already extensively analysed in the literature regarding their performance, scalability and flexibility. Therefore, throughout the research process, it was perceived that the objectives of this work should not focus on benchmarking those technologies, but rather integrate them in an architecture and prototype that leverages them to ensure the adequate collection, storage, processing and propagation of the data with characteristics such as variety, velocity and volume. The business requirements and indicators (2nd objective) were considered in the *Rules Repository* and the *Event Aggregator* components of the proposed architecture. Regarding the 3rd objective, the same is related with the processing of the data within the time frame needed for the several decision-makers in a specific context, and we can say that the same was not fully explored in this work. However, the simulations made in our demonstration case were considering data arriving at the system at each second, being considered an acceptable time frame to deal with the events in a real scenario. The 4th objective of this doctoral thesis (the inclusion of ML capabilities in the event processing) was considered in the proposed architecture, although it was not implemented in the demonstration case. Nevertheless, it was designed and defined how that component should be integrated in the *IEB* system, leaving as future work the technical and technological detail of the same along with adequate demonstration and validation. The 5th objective was related to the actions that should be performed in the system when some rule is verified. This topic was fulfilled with the definition of the *Triggers* component that is available as the connection to different destination systems performing the actions previously defined in the *IEB*. Within the demonstration case presented in Chapter 3, two different triggers were explored and successfully implemented: sending an e-mail to a Bosch collaborator and storing the data in a NoSQL database to be used in analytical applications (e.g., dashboards or real-time monitoring).

Chapter 4 was dedicated to the exploration of the first part of the *Mapping and Drill-down System* (the 6th objective of the doctoral thesis): a system that was designed to run in parallel with the *IEB* to monitor its growth once the same can be deployed in a complex context, with several processes producing data in a real time fashion and its monitoring can become a concern. This work was focused on the definition of a set of design rules that are used by the components responsible for inspecting the *IEB* code and was also focused on ensuring that all the *IEB* components are properly identified for future monitoring and logging of the system while running. In the demonstration case context and to validate the defined design rules, a flowchart reflecting the application of the design rules on the previously presented *IEB* system was defined, and then compared with the data collected physically on the code of the system (implemented for the demonstration case of Chapter 3). That validation proved that the rules, when applied correctly over the code of the system, can collect all the data needed to monitor the *IEB*. More than that, in this chapter, it is also presented some preliminary work on the Code Inspector and Logger component for the monitoring system, highlighting adequate expectations for the implementation of the *Mapping and Drill-down System*.

Chapter 5 presented the evolution of the definition of the Monitoring and Drill-down System, ensuring the adequate monitoring of the *IEB*. With the system designed in this work and the implementation of the demonstration case over some of the components of the *IEB* (producers and consumers), it was showed the appropriate monitoring of the system, collecting the data needed from the implemented components of the *IEB* and from the logging of the components that were running alongside the *IEB* system. In this work, a perspective over the metadata of the system and the data flowing within the system was presented in the Web Visualization Platform, after being stored and processed by the adequate technologies discussed in section 5.3.

Chapter 6 of this doctoral thesis is related to the Web Visualization Platform for the *Mapping and Drill-down System*. This work presented the proposal for the design and implementation of the Web Visualization Platform considering the challenges of adequate data visualization in Big Data contexts. In this sense, this platform was divided into key areas, where innovative techniques were used to create the most relevant and immersive visualizations for the *IEB Mapping and Drill-down System*. A demonstration case was developed in the context of Bosch to validate the architecture proposed for the Web Visualization Platform, as well as all the technologies suitable to implement the components of that architecture, so that their usage was better contextualized and justified. Regarding the demonstration case, the presentation of the platform to Bosch was considered a success, bringing significantly positive feedback from the company.

In summary, as mentioned above, all the objectives of this doctoral thesis were achieved or partially achieved (being these last ones explored in the Future Work section). With this doctoral thesis, the contributions for the scientific and practitioners' community were related with the gaps found in the literature. The *IEB* architecture proposed in this work presented in detail several components that can be used to build CEP systems in the era of Big Data, explaining how they should be related and communicate with each other. Moreover, the architecture considered and briefly presented how one can integrate a component that allows the inclusion of predictions and recommendations inside the rules' verification over the events flowing in the system. The *Mapping and Drill-down System* was significantly explored and showed how a complete monitoring system of the *IEB*, running in parallel to it, should be integrated, and why the same is relevant to be considered in this context.

7.2 Future Work

As mentioned in the previous section, all the main objectives of this doctoral thesis were achieved or partially achieved, as some of them were only specified and not as fully explored as the other ones. We can conclude that this fact was related with the time constraints for the doctoral thesis, the ambition of certain objectives, and the reality that some decisions needed to be made to give priority to other objectives in contrast to the ones discussed in this section and considered for future work.

The 3rd objective was defined to ensure that the *IEB* system was capable of matching the business needs regarding the time frames for data processing. Despite the achievement of acceptable time frames for the demonstration case (e.g., 1 event arriving at the system per second), future work can be considered together with Bosch to discuss and agree on latency requirements that would be then extensively benchmarked when the system is running. Considering this, we believe that throughput and latency will not be severe bottlenecks for the *IEB* system, as it relies on scalable Big Data technologies like Kafka and Spark to ensure the event collection, processing and consumption.

The 4th objective aimed to provide predictions and recommendations for the organization's daily activities. This topic was considered and briefly described in the *IEB* architecture presented in Chapter 3, but it was not explored in detail or integrated in the demonstration case, needing further work. In this context, a specially tailored architecture for this component can be designed in the future, explaining in detail which components will exist and how they will communicate with the Lake of Machine Learning Models where the previous trained algorithms will be stored, also providing the detail on the integration of those models with the rules' verification.

7.3 Research Publications

This doctoral thesis integrates in its chapters the 5 main scientific contributions of this work. Besides, some complementary contributions were also presented in conferences or published in journals or book chapters as specified below:

- Direct contributions for this doctoral thesis chapters:
    - 2 publications at CORE Ranking A Conferences
    - 3 publications at ERA/CORE Ranking B Conferences
- Complementary contributions for this doctoral thesis:
    - 1 publication at Scimago Q1 Journals
    - 1 publication at CORE Ranking A Conferences
    - 1 publication at ERA/CORE Ranking B Conferences
    - 2 publications at CORE Ranking C Conferences
    - 1 publication in a Springer Book

It should be considered that the number of publications enumerated above is higher than the number of chapters in this thesis addressing the main objectives and contributions of this doctoral thesis, as some of the papers were developed to explore the state of the industry 4.0 concept or some technologies to be considered for the *IEB* architecture. In this context, some of the papers were relevant to conduct the research work associated with this doctoral thesis but were not core components or contributions to achieve the objectives defined in this doctoral thesis. All the papers published within the scope of this doctoral thesis are enumerated below, considering an inverse chronological order:

- Direct contributions for this doctoral thesis chapters:
  - Conference Proceedings
    1. Andrade, C., Cardoso, M., Costa, C., & Santos, M. Y. (2021). Designing Monitoring Systems for Complex Event Processing in Big Data Contexts. European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS), Dubai.
       - ERA Ranking[22] B Conference
       - https://doi.org/10.1007/978-3-030-95947-0_2
    2. Andrade, C., Cardoso, M., Costa, C., & Santos, M. Y. (2020). An Inspection and Logging System for Complex Event Processing in Bosch's Industry 4.0 Movement. European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS), Dubai.
       - ERA Ranking[23] B Conference
       - https://doi.org/10.1007/978-3-030-63396-7_4

[22,23] Since the EMCIS conference is not included in the CORE ranking since 2020 for not being considered in the Computer Science area, we used the ERA ranking for the papers published in or after 2020.

3. Rebelo, J., Andrade, C., Costa, C., & Santos, M. Y. (2019). An Immersive Web Visualization Platform for a Big Data Context in Bosch's Industry 4.0 Movement. European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS), Dubai.
    - Best Technical Paper Award
    - CORE Ranking B Conference
    - https://doi.org/10.1007/978-3-030-44322-1_6
4. Andrade, C., Correia, J., Costa, C., & Santos, M. Y. (2019). Intelligent Event Broker: A Complex Event Processing System in Big Data Contexts. AMCIS 2019 Proceedings. Americas Conference on Information Systems, Cancún, Mexico.
    - CORE Ranking A Conference
    - https://aisel.aisnet.org/amcis2019/data_science_analytics_for_decision_support/data_science_analytics_for_decision_support/34/
5. Andrade, C. (2019). A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing. Proceedings of the Doctoral Consortium Papers Presented at the 31st International Conference on Advanced Information Systems Engineering (CAiSE 2019). CAiSE 2019, Rome, Italy.
    - Doctoral Consortium of a CORE A Conference
    - http://ceur-ws.org/Vol-2370/paper-01.pdf

- Complementary contributions for this doctoral thesis:
  - Conference Proceedings
    1. Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Pastor, O., & Marcén, A. C. (2019). Enhancing Big Data Warehousing for Efficient, Integrated and Advanced Analytics. 31st. International Conference on Advanced Information Systems Engineering (CAiSE'2019), C. Cappiello and M. Ruiz (Eds.): CAiSE Forum Papers, LNBIP 350, Rome, Italy, 3-7 June 2019, pp. 215-226
       - CORE Ranking A Conference
       - https://doi.org/10.1007/978-3-030-21297-1_19
    2. Correia, J., Santos, M. Y., Costa, C., & Andrade, C. (2018, September). Fast Online Analytical Processing for Big Data Warehousing. International Conference on Intelligent Systems, Madeira Island, Portugal.
       - CORE Ranking C Conference
       - https://doi.org/10.1109/is.2018.8710583
    3. Santos, M. Y., Costa, C., Galvão, J., Andrade, C., Martinho, B. A., Lima, F. V., & Costa, E. (2017). Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware. Proceedings of the 21st International Database Engineering & Applications Symposium, 242–252. New York, NY, USA: Association for Computing Machinery.
       - CORE Ranking B Conference
       - https://doi.org/10.1145/3105831.3105842

4. Santos, M. Y., Sá, J. O. e, Costa, C., Galvão, J., Andrade, C., Martinho, B., ... Costa, E. (2017). A Big Data Analytics Architecture for Industry 4.0. World Conference on Information Systems and Technologies.
    - CORE Ranking C Conference
    - https://doi.org/10.1007/978-3-319-56538-5_19

- Book Chapters

1. Costa, C., Andrade, C., & Santos, M. Y. (2018). Big Data Warehouses for Smart Industries. In S. Sakr & A. Zomaya (Eds.), Encyclopedia of Big Data Technologies (pp. 1–11). Springer International Publishing.
    - Springer International Publications
    - https://doi.org/10.1007/978-3-319-63962-8_204-1

- Journals

1. Santos, M. Y., Oliveira e Sá, J., Andrade, C., Vale Lima, F., Costa, E., Costa, C., ... Galvão, J. (2017). A Big Data system supporting Bosch Braga Industry 4.0 strategy. International Journal of Information Management, 37(6), pp. 750-760, 2017
    - Scimago Q1 Journal
    - https://doi.org/10.1016/j.ijinfomgt.2017.07.012

References

Agrawal, R., Kadadi, A., Dai, X., & Andres, F. (2015). Challenges and opportunities with big data visualization. *Proceedings of the 7th International Conference on Management of Computational and Collective IntElligence in Digital EcoSystems - MEDES '15*, 169–173. Caraguatatuba, Brazil: ACM Press. https://doi.org/10.1145/2857218.2857256

Andrade, C. (2019). A Big Data Perspective on Cyber-Physical Systems for Industry 4.0: Modernizing and Scaling Complex Event Processing. *Proceedings of the Doctoral Consortium Papers Presented at the 31st International Conference on Advanced Information Systems Engineering (CAiSE 2019)*. Presented at the CAiSE 2019, Rome, Italy. Retrieved from https://www.researchgate.net/publication/337740627_A_Big_Data_Perspective_on_Cyber-Physical_Systems_for_Industry_40_Modernizing_and_Scaling_Complex_Event_Processing

Andrade, C., Cardoso, M., Costa, C., & Santos, M. Y. (2020). An Inspection and Logging System for Complex Event Processing in Bosch's Industry 4.0 Movement. In M. Themistocleous, M. Papadaki, & M. M. Kamal (Eds.), *Information Systems* (pp. 49–62). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-63396-7_4

Andrade, C., Cardoso, M., Costa, C., & Santos, M. Y. (2021). *Designing Monitoring Systems for Complex Event Processing in Big Data Contexts* (M. Themistocleous & M. Papadaki, Eds.). In (pp. 17–30). Dubai: Springer International Publishing. https://doi.org/10.1007/978-3-030-95947-0_2

Andrade, C., Correia, J., Costa, C., & Santos, M. Y. (2019). Intelligent Event Broker: A Complex Event Processing System in Big Data Contexts. *AMCIS 2019 Proceedings*. Presented at the Americas Conference on Information Systems, Cancun.

Apache Kafka. (2018). Retrieved July 3, 2018, from Apache Kafka Homepage website: https://kafka.apache.org/

Babiceanu, R. F., & 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* (pp. 165–173). Springer International Publishing Switzerland 2015. https://doi.org/10.1007/978-3-319-15159-5_16

Bostock, M. (n.d.). D3.js—Data-Driven Documents. Retrieved July 30, 2019, from https://d3js.org/

Butcher, P. W. S., & Ritsos, P. D. (2017). Building Immersive Data Visualizations for the Web. *2017 International Conference on Cyberworlds (CW)*, 142–145. Chester: IEEE. https://doi.org/10.1109/CW.2017.11

Butscher, S., Hubenschmid, S., Müller, J., Fuchs, J., & Reiterer, H. (2018). Clusters, Trends, and Outliers: How Immersive Technologies Can Facilitate the Collaborative Analysis of Multidimensional Data. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, 1–12. Montreal QC, Canada: ACM Press. https://doi.org/10.1145/3173574.3173664

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

Chandler, T., Cordeil, M., Czauderna, T., Dwyer, T., Glowacki, J., Goncu, C., ... Wilson, E. (2015). Immersive Analytics. *2015 Big Data Visual Analytics (BDVA)*, 1–8. Hobart, Australia: IEEE. https://doi.org/10.1109/BDVA.2015.7314296

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

Costa, C., & Santos, M. Y. (2018). Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems. In J. Krogstie & H. A. Reijers (Eds.), *Advanced Information Systems Engineering* (pp. 459–473). Springer International Publishing.

Cugola, G., & 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

Ćurin, T., Bogadi, D., Volarević, M., Štajcer, M., Mihalić, A., & Mock, M. (2016). *Final Application Scenarios and Description of Test Environment* (No. D 1.2; p. 34). Hrvatski Telekom.

Dickey, D. A., Dorter, B. S., German, J. M., Madore, B. D., Piper, M. W., & Zenarosa, G. L. (2011). *Evaluating Java PathFinder on Log4J*.

Donalek, C., Djorgovski, S. G., Davidoff, S., Cioc, A., Wang, A., Longo, G., ... Drake, A. (2014). Immersive and Collaborative Data Visualization Using Virtual Reality Platforms. *2014 IEEE International Conference on Big Data (Big Data)*, 609–614. https://doi.org/10.1109/BigData.2014.7004282

Drools—Business Rules Management System. (2018). Retrieved October 13, 2018, from Drools Homepage website: https://www.drools.org/

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

ElSayed, N. A. M., Thomas, B. H., Marriott, K., Piantadosi, J., & Smith, R. T. (2016). Situated Analytics: Demonstrating immersive analytical tools with Augmented Reality. *Journal of Visual Languages & Computing*, *36*, 13–23. https://doi.org/10.1016/j.jvlc.2016.07.006

Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., & Mock, M. (2017). Issues in complex event processing: Status and prospects in the Big Data era. *Journal of Systems and Software*, *127*, 217–236. Scopus. https://doi.org/10.1016/j.jss.2016.06.011

Flouris, Ioannis, Manikaki, V., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Mock, M., ... Curin, T. (2016). FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms. *Proceedings of the 2016 International Conference on Management of Data*, 2093–2096. New York, NY, USA: ACM. https://doi.org/10.1145/2882903.2899395

Ganta, M. (2015, May 11). How-to: Build a Complex Event Processing App on Apache Spark and Drools [Blog]. Retrieved March 15, 2019, from Cloudera Engineering Blog website: https://blog.cloudera.com/blog/2015/11/how-to-build-a-complex-event-processing-app-on-apache-spark-and-drools/

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

Gorodov, E. Y., & Gubarev, V. V. (2013). Analytical Review of Data Visualization Methods in Application to Big Data [Research article]. https://doi.org/10.1155/2013/969458

Gupta, S. (2003). Introduction to Application Logging. In S. Gupta (Ed.), *Logging in Java with the JDK 1.4 Logging API and Apache log4j* (pp. 1–9). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4302-0765-8_1

Hadar, E. (2016). BIDCEP: A vision of big data complex event processing for near real time data streaming position paper - A practitioner view. *CEUR Workshop Proceedings*, *1600*. Scopus. Retrieved from Scopus.

Hermann, M., Pentek, T., & Otto, B. (2016). Design Principles for Industrie 4.0 Scenarios. *49th Hawaii International Conference on System Sciences (HICSS)*, 3928–3937. IEEE. https://doi.org/10.1109/HICSS.2016.488

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.*, *28*(1), 75–105.

Jayan, K., & Rajan, A. K. (2014). Sys-log classifier for Complex Event Processing system in network security. *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2031–2035. https://doi.org/10.1109/ICACCI.2014.6968471

Jazdi, N. (2014). Cyber physical systems in the context of Industry 4.0. *2014 IEEE Automation, Quality and Testing, Robotics*, 2–4. https://doi.org/10.1109/AQTR.2014.6857843

Jha, S., Jha, M., O'Brien, L., & Singh, P. K. (2016). Architecture for Complex Event Processing Using Open Source Technologies. *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, 218–225. https://doi.org/10.1109/APWC-on-CSE.2016.044

Kagermann, H., Wahlster, W., & Helbig, J. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group* (p. 82). acatech - National Academy of Science and Engineering.

Kim, N. W., Schweickart, E., Liu, Z., Dontcheva, M., Li, W., Popovic, J., & Pfister, H. (2017). Data-Driven Guides: Supporting Expressive Design for Information Graphics. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 491–500. https://doi.org/10.1109/TVCG.2016.2598620

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

Kumar Kaliyar, R. (2015). Graph databases: A survey. Communication & Automation International Conference on Computing, 785–790. https://doi.org/10.1109/CCAA.2015.7148480

Lan, L., Shi, R., Wang, B., Zhang, L., & Jiang, N. (2019). A Universal Complex Event Processing Mechanism Based on Edge Computing for Internet of Things Real-Time Monitoring. *IEEE Access*, *7*, 101865–101878. https://doi.org/10.1109/ACCESS.2019.2930313

Leavitt, N. (2009). Complex-Event Processing Poised for Growth. *Computer*, *42*(4), 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., & Vera, J. (1995). An event-based architecture definition language. *IEEE Transactions on Software Engineering*, *21*(9), 717–734. https://doi.org/10.1109/32.464548

Mei, H., Ma, Y., Wei, Y., & Chen, W. (2018). The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, *44*, 120–132. https://doi.org/10.1016/j.jvlc.2017.10.001

Miranskyy, A., Hamou-Lhadj, A., Cialini, E., & Larsson, A. (2016). Operational-Log Analysis for Big Data Systems: Challenges and Solutions. *IEEE Software*, *33*(02), 52–59. https://doi.org/10.1109/MS.2016.33

Nguyen, F., & Pitner, T. (2012). Information system monitoring and notifications using complex event processing. *Proceedings of the Fifth Balkan Conference in Informatics*, 211–216. Novi Sad, Serbia: Association for Computing Machinery. https://doi.org/10.1145/2371316.2371358

Nguyen, H. T. C., Lee, S., Kim, J., Ko, J., & Comuzzi, M. (2019). Autoencoders for improving quality of process event logs. *Expert Systems with Applications*, *131*, 132–147. https://doi.org/10.1016/j.eswa.2019.04.052

Oliner, A., Ganapathi, A., & Xu, W. (2012). Advances and challenges in log analysis. *Communications of the ACM*, *55*(2), 55–61. https://doi.org/10.1145/2076450.2076466

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

Pfeiffer, S. (2017). The Vision of “Industrie 4.0” in the Making—A Case of Future Told, Tamed, and Traded. *NanoEthics*, *11*(1), 107–121. https://doi.org/10.1007/s11569-016-0280-3

Pyne, S., Rao, B. L. S. P., & Rao, S. B. (Eds.). (2016). *Big Data Analytics*. New Delhi: Springer India. https://doi.org/10.1007/978-81-322-3628-3

Rebelo, J., Andrade, C., Costa, C., & Santos, M. Y. (2019, December). *An Immersive Web Visualization Platform for a Big Data Context in Bosch’s Industry 4.0 Movement*. Presented at the European, Mediterranean and Middle Eastern Conference on Information Systems (EMCIS), Dubai.

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

Shahzad, F., Sheltami, T. R., Shakshuki, E. M., & Shaikh, O. (2016). A Review of Latest Web Tools and Libraries for State-of-the-art Visualization. *Procedia Computer Science*, *98*, 100–106. https://doi.org/10.1016/j.procs.2016.09.017

Sicat, R., Li, J., Choi, J., Cordeil, M., Jeong, W.-K., Bach, B., & Pfister, H. (2019). DXR: A Toolkit for Building Immersive Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 715–725. https://doi.org/10.1109/TVCG.2018.2865152

Tableau. (2018). Tableau—Business Intelligence and Analytics Software. Retrieved October 17, 2018, from https://www.tableau.com/

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

Tsai, C.-W., Lai, C.-F., Chao, H.-C., & Vasilakos, A. V. (2015). Big data analytics: A survey. *Journal of Big Data*, *2*(1). https://doi.org/10.1186/s40537-015-0030-3

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., & Ganguli, D. (2014). Druid: A real-time analytical data store. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 157–168. Utah, USA: ACM.

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