Implementing a Monitoring Assessment Event Reasoning Software System Deployed on Kubernetes

Emmanouil Chatzimpyrros

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Electrical and Computer Engineering

Technical University of Crete
School of Electrical & Computer Engineering
Technical University of Crete
Akrotiri Campus, 73100 Chania, Crete, Greece

Thesis Advisor:
Associate Prof. *Sotiris Ioannidis*

This work has been performed at the Technical University of Crete, School of Electrical & Sciences Engineering, Technical University Department.

This work was partially supported by multiple European Projects, funded by the European Commission. This thesis reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING

Implementing a Monitoring Assessment Event Reasoning Software System Deployed on Kubernetes

Thesis submitted by
Emmanouil Chatzimpyrros
in partial fulfillment of the requirements for the
Masters' of Science in Electrical & Computer Engineering

THESIS APPROVAL

Author: Emmanouil Chatzimpyrros

Committee: Sotiris Ioannidis
Associate Professor, Thesis Supervisor, Committee Member

George Spanoudakis
Professor of Software Engineering, City University of London

Apostolos Dollas
Associate Professor, Committee Member

Chania, December 2022

Implementing a Monitoring Assessment Event Reasoning Software System Deployed on Kubernetes

Abstract

During the last years, the contemporary cyber systems, in their majority, are crowning security as the primary concern. As data collection has become more sophisticated over the last decade, organizations that collect and hold Personal Identifiable Information (PII) or any form of sensitive data are obliged by the relevant data protection legislation to provide guarantees that this data is handled in a certain, manner. Furthermore, organizations employing data need to be in line with the common security policies, dictated by the international/local standards, and must act proactively by following the best practices and taking measures to ensure the data sustainability as well as the successful detection of any type of cyber-attack.

Monitoring assessments are used to check for violations of security and dependability properties, which are necessary for the correct operation of the security solutions that are implied in a system. Events compose the necessary parts for these types of assessments, providing the fundamental abstraction for representing time-evolving information that may affect situations under certain circumstances. The research domain of complex event recognition and reasoning, focuses on tracking and analyzing streams of events in order to detect event patterns of special significance. The event streams may originate from various sources, such as sensors, computer networks, system log-files, video captors, etc. Additionally, the event stream's velocity and volume pose significant challenges to the event processing systems.

The aim of this thesis is to report a 'from-scratch-implementation' of a scalable runtime tool that serves this security assessment procedure followed up by a performance evaluation analysis. It is based on Everest, a logical reasoning system that provides event recognition and evaluation. It employs the Event Calculus formalism using Business Rules Management Language for the logical operations that a security policy assessment indicates. It also operates on a Kubernetes clustered based architecture for scalable and distributed event recognition for the core part, combining it with cloud deployment compliancy that is crucial for introducing it as a microservice.

Υλοποίηση Ενός Συστήματος Παρακολούθησης Αξιοπιστίας Συμβάντων με Λογισμό με την Χρήση Κυβερνετες

Περίληψη

Τα περισσότερα από τα υπάρχοντα συστήματα στον κυβερνοχώρο δίνουν προτεραιότητα στην ασφάλεια ως βασικό μέλημα. Ειδικότερα, καθώς η συλλογή δεδομένων γίνεται ολοένα και πιο περίπλοκη τα τελευταία χρόνια, οι οργανισμοί που συλλέγουν και διατηρούν προσωπικές πληροφορίες ταυτοποίησης (PII) ή οποιαδήποτε μορφή ευαίσθητων δεδομένων υποχρεούνται από τη νομοθεσία να παρέχουν εγγυήσεις ότι τα δεδομένα αυτά αντιμετωπίζονται με συγκεκριμένο τρόπο. Επίσης, οι οργανισμοί πρέπει να ευθυγραμμίζονται με τις κοινές πολιτικές ασφαλείας που υπαγορεύονται από τα διεθνή/τοπικά πρότυπα και πρέπει να ενεργούν προληπτικά, ακολουθώντας τις βέλτιστες πρακτικές, λαμβάνοντας μέτρα για τη διασφάλιση της βιωσιμότητας των δεδομένων, καθώς και για τον εντοπισμό κάθε είδους κυβερνοεπίθεσης.

Οι αξιολογήσεις παρακολούθησης χρησιμοποιούνται για τον έλεγχο παραβιάσεων των ιδιοτήτων ασφάλειας και αξιοπιστίας που είναι απαραίτητες για τη σωστή λειτουργία των λύσεων ασφαλείας που εφαρμόζονται σε ένα σύστημα. Τα συμβάντα συνθέτουν τα απαραίτητα κομμάτια για αυτούς τους τύπους αξιολογήσεων παρέχοντας τη θεμελιώδη αφαίρεση για την αναπαράσταση πληροφοριών που εξελίσσονται χρόνικα και που μπορεί να επηρεάσουν καταστάσεις υπό ορισμένες συνθήκες. Ο ερευνητικός τομέας της αναγνώρισης και συλλογισμού περίπλοκων γεγονότων, εστιάζει στην παρακολούθηση και ανάλυση ροών γεγονότων για τον εντοπισμό μοτίβων γεγονότων ιδιαίτερης σημασίας. Οι ροές συμβάντων μπορεί να προέρχονται από διάφορες πηγές, όπως αισθητήρες, δίκτυα υπολογιστών, αρχεία καταγραφής συστήματος, καταγραφείς βίντεο κ.λπ. Επιπλέον, η ταχύτητα και ο όγκος ροής συμβάντων θέτουν σημαντικές προκλήσεις στα συστήματα επεξεργασίας συμβάντων.

Ο σκοπός αυτής της διπλωματικής εργασίας είναι να αναφέρει μια από την αρχή υλοποίηση ενός εργαλείου χρόνου εκτέλεσης που εξυπηρετεί αυτή τη διαδικασία αξιολόγησης ασφάλειας. Το EVEREST είναι ένα σύστημα λογικής συλλογιστικής που παρέχει αναγνώριση και αξιολόγηση συμβάντων, χρησιμοποιεί τον φορμαλισμό λογισμού συμβάντων χρησιμοποιώντας τη γλώσσα διαχείρισης επιχειρηματικών κανόνων για τις λογικές λειτουργίες που υποδεικνύει μια αξιολόγηση πολιτικής ασφαλείας και λειτουργεί σε μια αρχιτεκτονική βασισμένη σε Kubernetes για επεκτάσιμη και κατανεμημένη αναγνώριση συμβάντων για το βασικό μέρος, συνδυάζοντάς το με τη συμμόρφωση με την ανάπτυξη με τεχνολογία νέφους που είναι ζωτικής σημασίας για την εισαγωγή του ως microservice.

Acknowledgments

I wish to express my sincere appreciation and gratitude to my supervisor, Professor Sotiris Ioannidis, for his convinced guidance, trust and encouragement for all these years that we work together from DiSCs years till now.

Also, I would like to express my gratitude for Professor George Spanoudakis and Dr. George Hatzivasilis that guided me through technologies and helped me to understand the wide spectrum of the logic particularly for this thesis.

Furthermore, I am grateful to Grigoris Kalogiannis, Dimitrios Deyannis and all the team members of Sphynx Technology Solutions AG that provided me with the will to extend my knowledge, practicing it both on industrial and academic level.

I would like to thank Professor Apostolos Dollas for his contribution as a member of the steering committee and for his helpful and instructional comments.

I would like to acknowledge the contribution of my family and my computer science friends for wisely counseling me throughout the years and selflessly supporting me whenever needed, especially my sister, who pushed me when I needed it the most.

Last but not least, I owe a special thanks, to a person that I will never see again, that supported me during all the uncertain times and stand by me for all life changing decisions, almost till the end.

Contents

List of Tables

List of Listings

List of Figures

Chapter 1

Introduction

In the abstract ecosystem of cybersecurity, ensuring the security and dependability of complex systems, operating in highly distributed environments and frequently changing contexts, is critical. One of the biggest challenges, alongside security and dependability assurance, is to maintain the systems' interoperability and adaptability. To achieve the latter, if a system is somehow compromised in any manner, there must be a dynamic way of reaction, by adapting or replacing some of its security mechanisms. Moreover, the owners of a system should have the ability to extract reports focused on security policies that are enforced and should be aware of blind spots or risk inducing behaviors in order to preserve the desired security and dependability. The distribution of the components in a modern, complex system, as well as the communications over heterogeneous and changing networks provide for an environment that is prominent to these security flows. Consequently, it raises the necessity of the consistent security and dependability reporting as top tier priority for system owners.

Considering the above mentioned facts, the key objective of the monitoring assessment tool that thesis is presenting has been the creation of a distributed runtime framework that works as a service, by enabling systems which operate in dynamic environments to configure, deploy and adapt mechanisms for realizing Security & Design (hereinafter S&D) Properties dynamically by assessing their components. Adding the notion of a cloud deployment, we are meant to have a holistic approach by significantly reducing the resources and provide this framework as a microservice that can fit in any system desired. During the deployment of an S&D Pattern by an application, it is necessary to monitor whether the invariant conditions specified in the pattern are satisfied and take corrective actions if a violation of these conditions is identified. Relevant to the basic concepts, the article from Chess [1] explains how extended static checking can be used to improve computer security by identifying and preventing errors in software before it is deployed. Krotsiani [2], takes this a step further, presents a method for ensuring the security of cloud services using a model-driven approach and continuous monitoring, which is shown to be effective through a case study.

With the ability to monitor these conditions, EVEREST is available as a service and when an S&D Pattern is activated, it undertakes the responsibility for checking logic

conditions regarding the runtime operation of the components that implement the pattern. These conditions are specified within S&D Patterns by monitoring assessment rules expressed in Event Calculus Assertion (hereinafter ECAssertion), which is a temporal formal language based on Event Calculus. The need of a mathematically proven ground truth such as Event Calculus that this monitoring framework is based upon, is crucial for a strong exhaustive deductive reasoning of interpenetrating security patterns, which establish logical certainty, and not deducting results from empirical arguments or non-exhaustive inductive reasoning, which establish "reasonable expectation".
This thesis covers the conception, the methodology, the implementation and the analysis of the described above monitoring framework.

1.1 Contributions

The contributions of this work are:

- We design and implement EVEREST, a distributed system able to perform runtime monitoring assessments. EVEREST S&D patterns are bound by a strict mathematical model yet are abstract enough to encapsulate all the security concepts that are necessary to describe a wide frame ecosystem of components in a system

- EVEREST is implemented in a way that it can detect violations of monitoring rules against streams of runtime events, which are sent to it by different and distributed event sources, called the Event Captors.

- EVEREST is also implemented to perform the following core functionalities.

  (i) deduce information about the state of the system being monitored, by using assumptions about the behavior of a system and how runtime events may affect its state of prioritizing some patterns from others,

  (ii) detect potential violations of monitoring rules (known as threats), by estimating belief measures in the potential of occurrence of such violations, and

  (iii) perform diagnostic analysis using external tools, in order to identify whether the events that cause a violation are genuine or the result of a system fault or an attack.

- We deploy EVEREST in Kubernetes, a Production-Grade Container Orchestration application that provides a cross platform compatibility and cloud compliance, that also handles and scales services.

- We evaluate EVEREST with a series of micro- and macro-benchmarks as well as with three real world applications. The results indicate that EVEREST provides an average performance speedup of 12.2x to similar applications, compared to a similar integration of legacy EVEREST integration, which was implemented in JAVA native language.

1.2 Outline

The rest of this thesis is organized as followed.Chapter 2 presents the background on the basic terms of Assurance levels in the modern ecosystem. Furthermore, we explain how organisations acquire certifications and extend the literature by mentioning the assurance levels of compliance and the necessary concepts for certification in realistic scenarios In Chapter 3, we present the strict mathematical model used to create the core of EVEREST, Event Calculus, that encapsulates all the security concepts that may needed to perform runtime assessments. We provide also an extended overview of Drools, which is the tool used to port Event Calculus core concepts into a performant logical machinery and create a strong logical knowledge base for our application, by providing some examples of the syntax of the rules that our application receive as an input. In Chapter 4, we get in depth on the implementation flow by mentioning the technologies used and why they were chosen, how we port the Event Calculus on Drools and having an overview of the external components that subjoin to the integration of EVEREST as a solution. In Chapter 5, we describe extensively the main components of EVEREST, the external tools and its execution flow for the two main deployment versions that are created to cover the needs of different organisations. Chapter 6, presents an in-depth analysis and evaluation of EVEREST with a variety of micro- and macro-benchmarks and in Chapter 7, we analyze our future steps.

" ACHTUNG! Das machine is nicht fur gefingerpoken und mittengrabben. Ist easy schnappen der springenwerk, blowenfusen und corkenpoppen mit spitzensparken. Ist nicht fur gewerken by das dummkopfen. Das rubbernecken sightseeren keepen hands in das pockets. Relaxen und vatch das blinkenlights!"

blinkenlights

Chapter 2

Background

2.1 Cybersecurity and Assurance

Individual businesses, as well as Supply Chains (SCS), are recognized internationally and particularly by the European Union (EU) as key enablers for the economic growth; thus, the managerial capability is directly linked with the level of efficiency and effectiveness. Many businesses that are mainly a part of a Supply Chain outsource a variety of their processes, critical information, and Information Communication Technologies (ICT) services to third parties and highly interdependent dispersed nodes of heterogeneous cyber-physical infrastructures.

In this chapter we analyse the concepts of certifications and assurance as well as the existing general indications for the process of acquiring certification by organisations. An analysis of the extended concepts that are revolving Certification will not be extended thoughtfully, due to the fact that the legislative ecosystem is quite broad, and a legal analysis is not in order with the scope of the thesis.

2.2 Certification

In this sub-section we will provide a summary of the basic concepts of the Certification process, to outline the role of the EU interest mainly to regulate the spectrum of privacy and security measures that need to be imposed by an organization or by a group of organizations, by building an information safe, security resilient and trustworthy environment for businesses and service providers. As a result of this process, we also highlight the necessity of the tools that have crucial role to provide a complete cybersecurity posture of these organizations. This environment provides assurance not only as a regulatory measure for businesses; from the client's perspective, it supports the development of their confidence and trust to the services provided by an EU Certified organization.

The proposal of the NIS Directive 2.0 (NIS 2 Directive) [3] contains a series of measures for improving cybersecurity infrastructure and particularly the resilience and incident response capabilities of public and private competent authorities. It provides a variety of indications that encapsulate the overal cybersecurity posture of an organization. Also,

one of the key elements of the Commission's proposal is to address the security of supply chains and supplier relationships by requiring individual companies to address cybersecurity risks in supply chains and supplier relationships. Cybersecurity certification of the SCS can be considered as a mitigation action against cybersecurity SCS risks.

As the threat landscape is enormously evolving, the Regulation (EU) 2019/881 of the European Parliament and the Council, known as EU Cybersecurity Act (EUCSA) [4] promotes cybersecurity certification for ICT products (software, hardware, processes, services) and it will scale up the response to cyber-attacks, fostering cyber resilience and trust for consumers within the EU. The EUCSA provides the basis for the creation of the EU certification framework for ICT products; it provides a framework on standards, namely based ISO/IEC 15408 [5] , that advanced security assurance cases can be based on [6]. This framework is also known as Common Criteria (CC) and has been updated by various ISOs so far, such as ISO/IEC 18045 [7]. These ISOs can be easily coordinated in an integrated model of knowledge management for the security of information technologies, as presented in [8].

The EU cybersecurity certification is defined as a comprehensive set of rules, technical requirements, standards and procedures that are established at the Union level and that apply to the certification or Conformity Assessment (CA), which its importance is highlighted on a National level [9] and by Stephenson [10], well as of specific ICT products (software, hardware, systems, services). Each certification scheme specifies the categories of products and services covered; the cybersecurity requirements that need to be met -such as standards or technical specifications-, the type of evaluation that is planned to be done - such as a self-assessment or a third-party assessment - and the intended level of assurance that is going to be achieved. The certificates will be valid across all Member States (MSs).

These certification procedures, as well as the general guidelines, protect citizens of the EU from malicious activities and the leakage of PII, that are often used for extracting advertising profits. These principals are imposed to every organization that is active in EU soil or has expanded its services to the EU and they are expressed in the GDPR regulation [11], which dictates the significant changes that are imposed from the EU in terms of possess and handle information explained in Albrechts [12].

2.3 Acquiring certifications as a company

There are different Certifications that an organization may need to acquire (ISO, NIST, FIPS and much more). The EUCC scheme for example, is based upon Article 54 of the EUCSA [13]. The latter presents in detail the key elements that an EU certification scheme should include. By using the EUCC, any ICT product can serve as a Target of Evaluation (TOE) and can be subject to a security evaluation, also known as Conformity Assessment (CA), in which it is assessed against security requirements. The CA of the TOE is defined as the procedure that is followed for evaluating whether specified requirements relating to the TOE have been fulfilled. Throughout the CA process, the TOE should be identified, and security aspects should be concretely specified.

2.4 Common Criteria and Assurance Levels

The Common Criteria (CC) for Information Technology Security Evaluation, published jointly by the International Organization for Standardization and the International Electrotechnical Commission [14]. CC provide international guidelines for evaluating the security of IT systems, widely expressed in Fekete's [15]. The assurance requirements outlined in the CC, including prepackaged sets of Evaluation Assurance Levels (EALs), are based on the idea that more thorough evaluation leads to greater assurance of security as explained in [16]. The Assurance Levels are a general term that applies to every certification scheme, although they are all based on the CC. In general, the CC assures that the specification, implementation, and evaluation process of a computer security product has been conducted. To obtain assurance in CC, the analysis and the gathering of operational evidence phases need to be considered. The former includes:

- Traceability/coverage analysis
  o Between TOE design elements
  o Between TOE design elements and Security Functional Requirements (SFRs) [17]
- Analysis of functional tests coverage and results
- Analysis of vulnerabilities
- Verification of proofs whereas the latter, includes:
- Independent functional testing
- Penetration testing
- Checking that procedures have been applied

EAL refers to a ranking category assigned to an IT product or system after the CC evaluation. The levels that will be studied, indicate the extent to which this product or system was tested. In general, the CC includes seven levels of increased Assurance (described in 3 below) whose scale is based on the following aspects:

- The level of assurance obtained (low security to high security in very risky contexts).
- The use of analysis and operational evidence techniques of varying formality (informal, structured, semi-formal, formal).
- The varying requirements regarding the level and formality of evidence are provided for evaluation.
- The varying requirements regarding the TOE development process (no conditions, to tool-supported, and to formal and fully accountable development processes).

2.5 Necessary concepts for Certification

When the responsible for an organization's service or a Supply Chain Service Provider (SCS-P) wishes to get a SCS certificate or to maintain the certification of an already certified SCS, the provider shall usually submit a document, following the template defined in the Certification Scheme, regarding the content requirements. The latter, must be filled out with all the required information, which depends in part on the reason that triggered the CA. The process that organizations undergo to obtain certification can vary. During the evaluation, the organizations or the SCS shall submit all the information needed to demonstrate that the implementation of their SCS meets the security requirements defined for the targeted assurance level, including but not limited to:

- Policies and procedures that apply to the design and operation of the organizations/SCSs under evaluation;
- documentation related to the organizations/SCSs under evaluation, including the view adopted and elements of the view (e.g., in process view then the processes and business partners along with the Mutual Agreement needs to be provided);
- if required, records that can be used as evidence that the above-mentioned policies and procedures are being followed e.g., the individual security policies, the documentation of all controls of the organizations/SCS assets;
- if third party organizations/subcontractors are used, the business partners need to provide assurance for their security levels.
- where explicitly stated, specific documents and records required by the assessor to assess the fulfilment of requirements pertaining to specific security controls.

The acquisition of different levels of security certification on a company or a Supply Chain is a highly demanding process. In terms of applying and enduring the process, companies must be prepared adequately and act proactively by enforcing security policies based on the security properties that arise from the certification guidelines.

The EVEREST tool that the current thesis is presenting, can be used from a wide range of companies that plan to acquire certifications for cybersecurity properties. Specifically, this tool monitors those security properties that are set and produces a cybersecurity posture notion to ensure that these properties are well applied and interconnected with each other. It performs the former tasks by identifying the flaws in the applicability and by notifying the company about its cybersecurity status. After completing the process of conducting multiple assessments and reaching the requirements, the organizations may be ready for initiating the certification process.

2.6 Real world security property checkout

Covering a great spectrum of possibilities, monitoring and adaptation of system security and dependability runtime mechanisms are considered a solid solution for the needs of

different kinds of organizations. We focus more on the business ecosystems but we have to take in consideration that assurance, can be used also outside of a strict business model structure. The usability of a monitoring assessment that provides assurance for example, can outlined in Spanoudakis' [18] work on dynamically verifying the correctness of peer-to-peer systems, which are distributed networks that allow users to share resources and information directly with each other. In order to underline the need of a monitoring assessment tool in a modern organization that uses multiple servers and processes and has many interactions between its assets and the involved personnel, we must consider a realistic scenario.

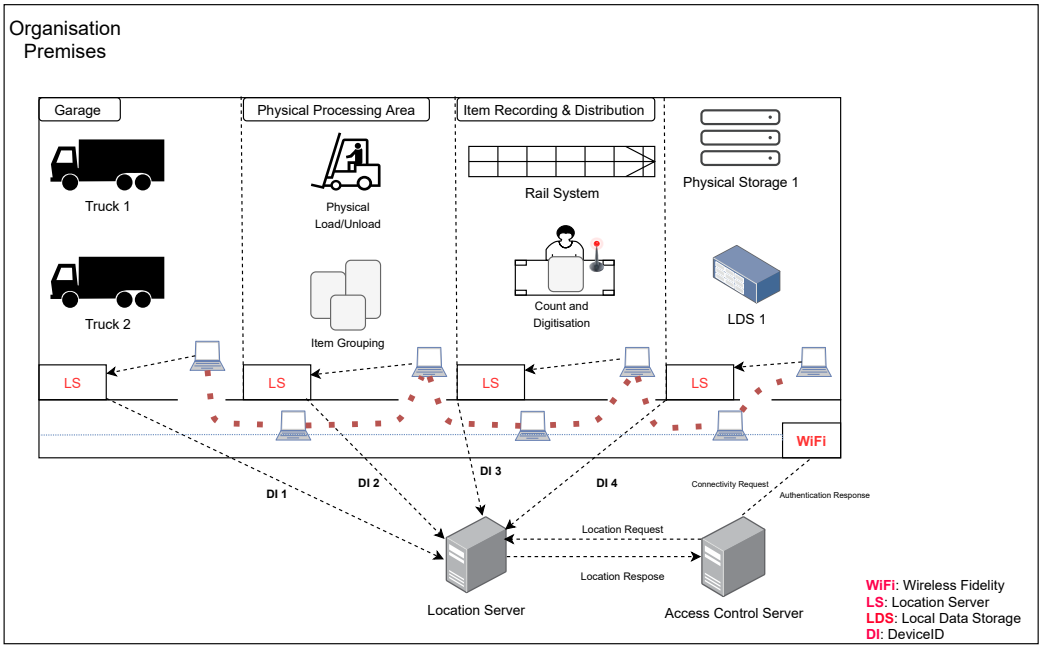

Figure 2.1: Location Based Access Control System example deployed in a modern organization.

Considering a modern, complicated system which complies with the modern realized world of the supply chain ecosystem ***Figure 2.1***. We describe and focus on a specific example of an organization that handles delivery and storage of various types of products and its responsibilities are expanding into both the digitization and physical storage of those products. As part of the organizations' software assets, the system handling the security policies should manage access to different resources of an organization by combining user authentication, device identification and location detection capabilities. In this system, referred to Location Based Access Control System (LBACS) in the following, users and managerial personnel, entering and moving within the premises of an organization using mobile computing devices (e.g., a notebook or a smart phone) may be given access

to different resources, such as the enterprise intranet, printers, different Internet of Things devices (IOT), process verificatory devices or the Internet, depending on their user-id, the id of the mobile device that they are using and the location of this device. We also elaborate into an example that is provided by the integrated native version of the monitoring assessment tool that is described in published work by Spanoudakis [19]. Resource access is granted depending on policies, which determine when access to a particular type of resource is considered to be harmful or not. We consider the view as for the simple process of collection-digitisation and storage of items that come from Trucks. As for the process itself in more detail we examine the following steps:

1. Items come through trucks that the company owns or is collaborating with. Note that the privileges on device access on the trucks that are owned by the company and the trucks that are contracted to external collaborating organizations may differ.

2. Trucks are unloaded by the responsible personnel and are grouped together in bunches. Every bunch is representing a number of items that came from a specific truck.

3. Grouped items are unloaded and stored in a categorized manner, holding the ID of the Truck they were unloaded from. Also, the digitization of the products (such as ItemID, TruckID, CategoryID, Sender, Receiver, Fragility) is completed.

4. Items are stored in a physical storage and the information of the stored items is being transmitted to the local data server and from there it is sent to the main Data Server.

This type of organization functions by enforcing a number of policies that must be followed by the personnel in all of these steps. The goal is to keep the productivity high and provide straightforward pipelines that must be executed by the personnel involved in any of these processes. A policy may, for example, determine that an authenticated managerial employee of the organization who is trying to access a process verification device via the local wireless network while being in an area of the premises that is accessible to the public, should be granted access. In the same manner, authenticated visitors or external employee personnel should only be given access to different devices when they are in one of the organization's different processing rooms.

As an addition, we consider an enhanced security policy that implies that every industrial section of the premises is responsible for completing a specified procedure. All the procedures must be completed in a serialized manner in order to manage the digitization of the products and the metadata that arise from each part must be recorded. For example, items from Truck 1 must be processed as a group, carrying this information until the physical and digital storage (e.g., which items come from which truck, from whom the item is processed and digitized etc). Consider a manager who verifies this serialized procedure using the software devices located in every room of the described organization, giving the final signal to the procedure carried out in a specific room. Also, we should keep in mind that every premises' room of this procedure, contains certain IoT devices that are

inaccessible from other rooms. Given this as a fact we can envision that there could be some kind of authorization rule that demands users be of a certain privilege, in order to have access in devices in different rooms. Per se, a Storage Officer could be eligible to move to the first two rooms (Garage and Physical Process Area) and access the devices in each room when moving, but she may not be eligible to access devices in the other two rooms as her privilege is limited.

The general architecture of this expanded process as well as the LBACS is displayed in. As shown in the ***Figure 2.1***, the access control solution of LBACS is based on two servers: a location and a control server. The control server polls the location server at regular intervals in order to obtain the position of the devices of all the users currently connected to the system. The location server calculates the position of different user devices from signals it receives from devices through location sensors. The estimates of device positions are not exact and are associated with an accuracy measure. The authentication of the identity of the different user devices is based on the existence of a TPM chip on them and its ability to respond to requests by the authentication server of the system.

The effectiveness of the access control solution of LBACS depends on several conditions regarding the operation of the different components that constitute it at runtime including:

1. The continuous availability of the location servers, TPMs on the user devices and main Data server at runtime. The availability of these components is a pre-requisite for the availability of the device position and the authentication information, which are necessary for the access control system at runtime.

2. The continuous periodic dispatch of signals from the mobile devices to the location server that enables it to maintain accurate position data for the devices.

3. Also, we can indicate as an extra step completed security solution, the prevention of data loss into the system, combining the Dataserver availability with an indication that disables send-service that the Local Datacenter is attempting and prevent data loss. Once the availability is back on the main Dataserver, we must re-enable the data send-service

Monitoring the above conditions at runtime in a system like LBACS would require the implementation of appropriate checks within the system itself or the deployment of an external monitor that would take on the relevant responsibility. The former option would not be very flexible as it would require changes in the implementation of the required checks when the different components of the system change.

Furthermore, depending on changes on the system components, the exact conditions that would need to be monitored could change as well. In such cases, giving the system the responsibility for monitoring would not be flexible. The solution advocated in monitoring assessment system is to delegate this responsibility to external components that would check the above conditions and take action when they are violated, e.g., replace malfunctioning components, alert system administrators of detected violations etc. In particular, in a monitoring assessment tool the responsibilities for monitoring runtime conditions

as well as the evaluation of them from the results perspective(satisfaction/violation of property), is assigned to our monitoring assessment framework which is the contribution presented on this thesis, namely EVEREST. The strong mathematical basis of the monitoring assessment tool that we are building gives as the opportunity of counter play the complexity and move it to the upper layer of the rule representation.

Chapter 3

Logical Core

3.1 Event Calculus

Event Calculus (EC) is a logic language for representing and reasoning about actions and their effects as time progresses expressed initiatly by Kowalski [20] who later expanded this work by presenting several variations of the Event Calculus, including first-order, second-order, and modal formulations, and discusses their relative advantages and relationships with one another in [21]. EC is a formalism for reasoning about action and change. The core EC concepts contain the following definitions:

- **actions** – which are called events and indicate changes in the environment
- **fluents** – which are time-varying properties (predicates/functions)
- **timepoint** sort – which implements a linear time structure on which actual events occur.

It is based on first-order predicate calculus and is capable of simulating a variety of phenomena such as actions with indirect effects, actions with non-deterministic effects, compound actions, concurrent actions, and continuous change. The EC defines predicates for expressing, among others, which fluents hold and when (*HoldsAt*), what events happen (*Happens*) and what their effects are (*Initiates*, *Terminates*). It adopts a straightforward solution to the frame problem which is robust and works in the presence of each of these phenomena.

In a generalized view, event calculus is working using a Logic Machinery to provide evaluations regarding actions that are happening or have happened in a past timeline. The Logic machinery can provide also representations of predicates, values that specific actions are meant to alter, in order to extract evaluations of aggregated actions upon demand.

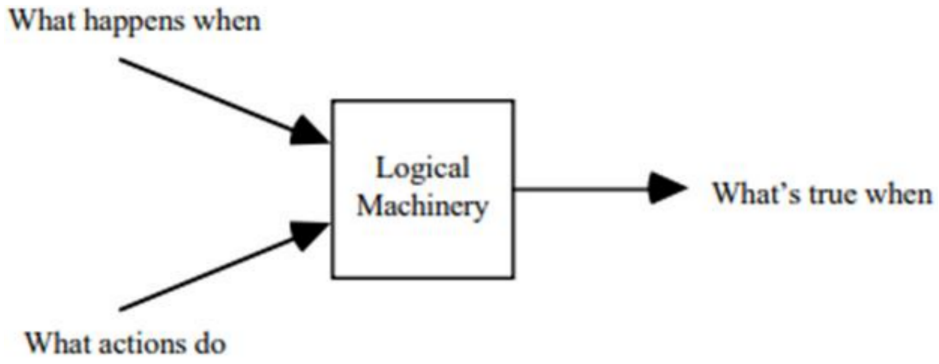

Figure 3.1: Event Calculus abstract logic representation.

Expanding this generalised approach, we can add the aforementioned properties described into the ***Figure 3.1*** and expand in terms of detail. Subsequently, the **Initially**, **Happens**, and temporal ordering formulae, as well as the **Initiates** and **Terminates** formulae, are passed through the Event Calculus Axioms. These actions produce the outcome represented as a fluent in ***Figure 3.2***, that holds the value produced, and we can either to re-enter a rule logic machinery or to produce the outcome fluent (True/False) that evaluates the sequences of actions performed.

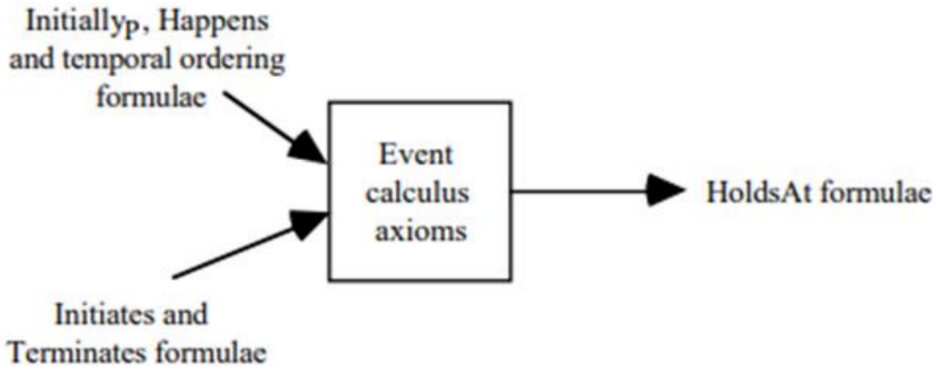

Figure 3.2: Expanding ***Figure 3.1*** concepts of Event Calculus.

As for the Event Calculus Axioms, were expanded in already published research by Shanahan [22]. The research is very important to the core of the Event Calculus and concludes the above-mentioned logic. In a generalized manner, if we see the predicates as the building blocks of this logical representation, the axioms could be seen as the concatenation that keeps these blocks interconnected. The following four axioms in ***Table 3.1***, illustrate the basic structure of Event Calculus as it can be defined.

For the first Axiom **(EC1)** we define Clipped predicate, which as we see is referring to a predicate that 'locks' the logical interpretation of a state, meaning that we cannot change the value of a fluent **(EC2)** when another value is meant to be terminated by a specific event (Happens). If we observe the bigger picture for these axioms, we can come into the

Table 3.1: Axioms in Event Calculus

EC(1)	$Clipped(t1, t, t2) <= (\exists e, t) Happens(e, t, \mathbb{R}(t1, t2)) \wedge Terminates(e, f, t)$
EC(2)	$HoldsAt(f, t) <= Initially(f) \wedge \neg Clipped(0, f, t)$
EC(3)	$HoldsAt(f, t) <= (\exists e1, t) Happens(e, t, \mathbb{R}(t1, t) \wedge Initiates(e, f, t1) \wedge \neg Clipped(t1, f, t)$
EC(4)	$Happens(e, t, \mathbb{R}(t1, t2)) => (t1 < t) \wedge (t \leq t2)$

conclusion that they are describing the process of changing a HoldsAt value (fluent); in this, we insert the logic that "a fluent cannot be changed when the fluent is in the process of Termination or Initiation **(EC3)** by a specific event that demands this alternation".

The EC supports context-sensitive effects of events, indirect effects, action preconditions, and the commonsense law of inertia [23]. Certain phenomena are addressed more naturally in the event calculus, including concurrent events, continuous time, continuous change, events with duration, nondeterministic effects, partially ordered events, and triggered events. Examples of such phenomena could be:

- The commonsense law of inertia: when moving a glass does not cause a glass in another room to move.
- Release from the commonsense law of inertia: if a person is holding a PDA (Personal Digital Assistant), then the location of the PDA is released from the commonsense law of inertia so that the location of the PDA is permitted to vary.
- Event ramifications or indirect effects of events: the PDA moves along with the person holding it (state constraint) or instantaneous propagation of interacting indirect effects, as in idealized electrical circuits (casual constraints).
- Conditional effects of events: the results of turning on a television depend on whether it is plugged in or not.
- Events with nondeterministic effects: when flipping a coin, it results in the coin landing either heads or tails.
- Gradual change: the changing height of a falling object or volume of a balloon in the process of inflation.

The EC contains a set of fluents, a set of events, and a partially ordered set of time points. In the EC, the description of the worlds (possible scenarios) is based on the following axiom (assume e is an event, **f** is a fluent, and **t**, **t1**, and **t2** are time points):

1. **Initiates**(e, f, t): **f** holds after event **e** at time **t**
2. **Terminates**(e, f, t): **f** does not hold after event **e** at time **t**
3. **InitiallyP**(f): **f** holds from time **0**

4. **InitiallyN**(f): **f** does not hold from time **0**
5. **Happens**(e, t1, t2): event **e** start at time **t1** and ends at **t2**
6. **HoldsAt**(f, t): **f** holds at time **t**
7. **Clipped**(t1, f, t2): **f** is terminated between **t1** and **t2**
8. **Declipped**(t1, f, t2): **f** is initiated between **t1** and **t2**

In EC, specific values of the fluents describe a state. An event which changes the value of one or more fluents has as a consequence the change of the state. An evolution of the world is a sequence of actions and states.

We formally define events and fluents. Events occur sequentially or in parallel on denoted time points. Events can change the state of fluents and trigger new events. These transactions are modelled by rules. The rules have preconditions. If the preconditions are satisfied (left-hand statements), the rule is executed and may change the state of a fluent. At this point, we accept statements that have been proven by the predicate calculus. According to these statements, we model rules that implement the management logic. A fire alarm can be modelled by an event. The event triggers a set of rules which check if there are any actions that must be taken (functional and non-functional properties of reaction strategy). When the counteractions are completed, another set of rules can be triggered to determine which SPD and safety properties are satisfied. The events and the changes they cause, produce a trace in time. The final state of the trace determines the final outcome. Examples for event calculus theoretical problems that can be found in Sergot (Sergot, 2006) and Mueller [24] show the logical representation of the aforementioned simplified basis in a real-world problem. Additionally, Liu [25] focusing on Self adaptation of multi-agent systems expands the requirements for self-adaptation; they are facilitating monitoring and reasoning about the actions of agents, achieving requirements-driven planning at runtime. The reasoning behaviour of Monitor is modelled in Event Calculus (EC). EC is implemented in Everest n Drools logical language, while some functionality is further implemented in Java. It is a first-order temporal logic which can both represent and reason actions and their effects over time. Abstracting the above concepts, the basic elements of EC are comprised by events and fluents.

We introduce EVEREST, which is a monitoring tool that performs continuous assessments and is based upon these core logic factors of EC that we mentioned, comprising the rules that are continuously checked in a system. An event in EC is specified as something that occurs at a specific instance of time and is of instantaneous duration. Furthermore, it may cause some change in the state of the reality being modelled, which is represented by fluents. The logic is implemented in the form of reasoning rules. Based on the ongoing events and the status of the related fluents, when the preconditions of a rule are satisfied (left side of the rule), the rule fires and performs some actions (right side of the rule). In our case, set of rules define monitoring criteria for the CIA or other security/privacy aspects of the examined assets. As a result, the core functionality of EVEREST introduces EC and its main elements as the basis of its logic.

Thereupon, the EVEREST reasons (using the rule sets/axioms) and maintains the status of the monitored assets (using fluents), and exchanges information with the other components (i.e., Beats and Event Captors, AutoML capabilities tool, Incident Response Tool and the Dashboard) based on **messages**, which **resemble events in the EC**. This information is also maintained in the knowledge base.

3.2 Drools

Drools is a collection of tools that allow us to separate and reason over logic and data found within business processes. Drools is a Production Rule System which is Turing complete [26], with a focus on knowledge representation to express propositional and first order logic in a concise, non-ambiguous and declarative manner. The brain of a Production Rules System is an Inference Engine that is able to scale to a large number of rules and facts. The Inference Engine matches facts and data against Production Rules - also called Productions or just Rules - to infer conclusions which result in actions. It is 100% integrated in Java and it is an open source project backed by JBoss and Red Hat.

Drools implements and extends the Rete algorithm. The Drools Rete implementation is called ReteOO, signifying that Drools has an enhanced and optimized implementation of the Rete algorithm for object-oriented systems [27]. Other Rete based engines also have marketing terms for their proprietary enhancements to Rete, like RetePlus (IBM, n.d.) and Rete III [28]. The most common enhancements are covered in Production Matching for Large Learning Systems Doorenbo's article [29]. There are many extensions of the RETE algorithm expressed in many previous works such as the extension of RETE through the concepts of time-stamped events and temporal constraints between events that allows applications to write rules that process both facts and events expressed by Berstel [30], and also extending in terms of rule decomposition in Alpha-Node-Hashing and BetaNode-Indexing in Liu's work [31].

The Rete algorithm can be broken into 2 parts: rule compilation and runtime execution. The compilation algorithm describes how the Rules in the Production Memory are processed to generate an efficient discrimination network. In non-technical terms, a discrimination network is used to filter data as it propagates through the network. Any Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The path from the root node to a leaf node defines a complete rule left-handside. Each node has a memory of facts which satisfy that pattern. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or a combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

The Rete algorithm implementation wise, is an efficient pattern matching algorithm for implementing production rule systems The Rete algorithm is designed to sacrifice memory for increased speed. The Rete algorithm exhibits the following important characteristics:

- It reduces or eliminates certain types of redundancy through the use of node sharing.

- It stores partial matches when performing joins between different fact types.
- It allows for efficient removal of memory elements when facts are retracted from working memory.

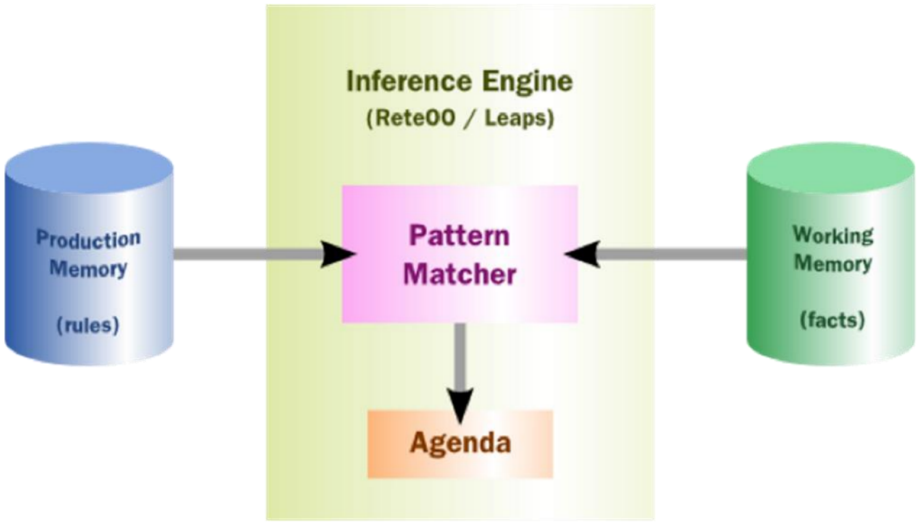

Figure 3.3: High-level View of a Production Rule System. Representing the Inference Engine of Drools.

Inference Engine is representing a computer program that tries to derive answers from a knowledge base existing in Production Memory and a Working Memory (rules and facts). The latter in our case is the core of the Monitoring System that we are building, which uses the Event Calculus rules to compile the pattern matcher and await events represented as facts, as shown in ***Figure 3.3***, to extract the results represented as Satisfactions or Violations of these rules. An extra addition of Drools in the classic Rete algorithm is the Conflict Resolution that is required when there are multiple rules on the agenda. As firing a rule may have side effects on the working memory, the rule engine needs to know in what order the rules should fire (for instance, firing rule A may cause rule B to be removed from the agenda) or in the case of the Monitoring Assessment tool, rule A could produce facts that feed rule B that produces an assessment result. The default conflict resolution strategies employed by Drools are Salience and LIFO (Last In, First Out).

There are two methods of execution for a rule system: Forward Chaining and Backward Chaining; the systems that implement both are called Hybrid Rule Systems. Drools is a forward chaining engine. Forward chaining is "data-driven" and thus reactive, with facts being asserted into working memory, which results in one or more rules being concurrently true and scheduled for execution by the Agenda. In short, we start with a fact; this fact then propagates and we end in a conclusion.

When defining the Drools semantics in the abstract Drools syntax, we explained that rules are pieces of knowledge often expressed as, "when some conditions occur, then do

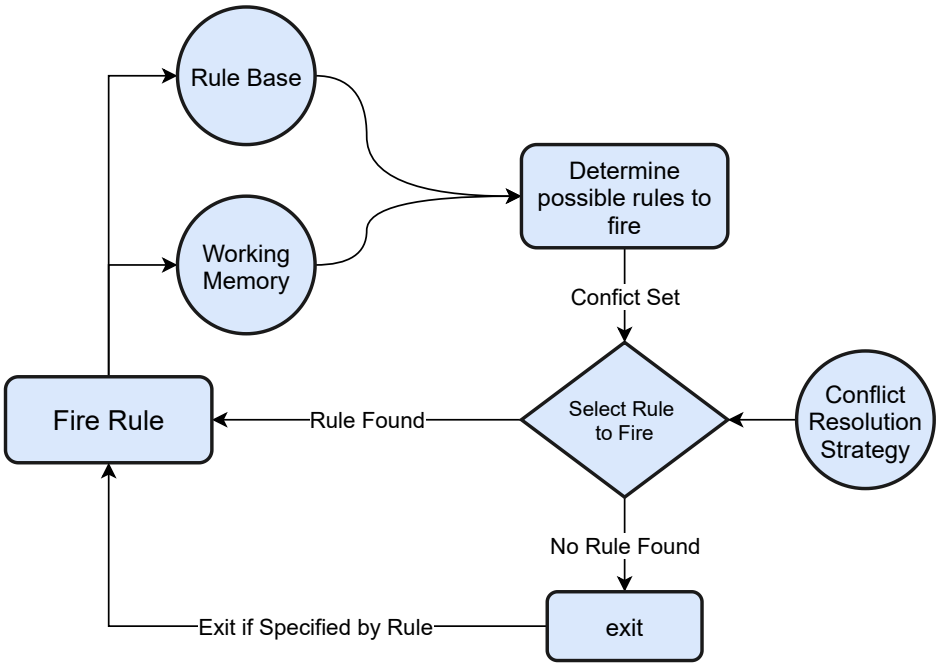

Figure 3.4: Drools Semantics for the reactive forward chaining and how the facts are managed by the system.

some actions" as represented in ***Figure 3.4***. This logic is represented in the following manner in ***Table 3.2***, having the Left Hand Side (LHS) and the Right Hand Side (RHS).

Table 3.2: Drools Logic

When (LHS)
<Conditions are true>
Then (RHS)
<Do specified actions>
End

The most important part of a Rule is its 'when' part. If the 'when' part is satisfied, the 'then' part is triggered as shown in ***Table 3.3***.

Drools introduces also global variables that can be used by all the rules. In a typical monitoring assessment, we introduce the event calculus logic into the 'when' part and on the 'then' part we introduce the storage of the assessment results as well as execution events and retracting memory Objects.

Drools also perform node sharing. Many rules repeat the same patterns, and node sharing allows us to collapse those patterns so that they don't have to be re-evaluated for every single instance. We use the same logic for our case to produce different results (Violation/Satisfaction). A detailed example will be analyzed in next chapters in order for

Table 3.3: Drools Rule Syntax

Rule_*<rule_name>*
<attribute><value>
When
<conditions>
Then
<actions>
end

all these terms to be encapsulated and understood fully.

3.2.1 Event Calculus in Drools Syntax

In order to encapsulate the concepts of event calculus already described into the drools logic engine, we needed to somehow port the axioms of event calculus into the Drools specified logic. The first step for this integration was to define a very well-defined object-oriented structure that holds the basic principles of event calculus in Drools syntax. The challenge in this position, was to syntax the drools itself in a manner that explains key concepts of event calculus such as the predicates (Happens, HoldsAt) and additionally define the EC Axioms.

Moreover, from a technical perspective, we had to keep in mind the memory safety while defining this common schema and provide solutions in axioms to balance the code coherence and the theoretical/logical concepts of the EC in Drools syntax. Similar kind of implementations are referred to the Cerbere which is a Jess tool production system designed to perform online causal, temporal and epistemic reasoning based on the Event Calculus [32]. Those terms are represented in the following contextual form and are mapped with the Axioms of event calculus that were described in previous chapter.

3.2.2 Drools Rule Syntax

By using an example to describe the conflict resolution capabilities, we can ideate two Rules that are correlated and give a priority to RuleA. So, in order to examine the salience property we consider a dummy example of a minified confidentiality property for user's IP in Drools that can be represented as in ***Table 3.4***.

In order to properly describe the context of this example, we define the whitelist array as a global variable that can be accessed from all the rules that are presented, containing some predefined IPs as values. This variable represents all the eligible IPs that a user can login from. Thus, if we now add an **A('63.43.80.92')** object into this logic session, the RuleA will first fire, evaluating that there is an Object-namely the A() Object-that has the IP and the username field that is required, so it will evaluate the condition as true and will continue to the *<action>* part of the RuleA. The 'then' part of RuleA rule will

Table 3.4: Drools With Event Calculus Example

```
Global String Array whitelist = ['8.8.8.8','63.43.80.92','195.32.45.12']

Rule RuleA salience 20
   when
      A($user, IP)
   then
      new B(loggedin=True, loggedip= IP)
end


Rule RuleB_Violation
   when
      A($user, IP)
      B(loggedin==True, loggedip == IP, whitelist not contains loggedip)
   then
      new Alert('User '+$user+'logged in from anauthorised ip!')
      retract(A,B)
end


Rule RuleB_Satisfaction
   when
      A($user, IP)
      B(loggedin==True, loggedip == IP, whitelist contains loggedip)
   then
      new Alert('User '+$user+'logged in from authorised ip.')
      retract(A,B)
end
```

create a new B() Object that contains the variables that are shown to the example (Boolean loggedin, String IP and String username).

Now, there is a B() object with certain values into the logic session, thus it will evaluate the second rule, RuleB_Satisfaction. We can see that the IP provided into the B() Object is not inquired into the global whitelist array of IPs, so we trigger an alert. That will highlight in the monitoring assessment result that there was another user logged in that did not comply with our confidentiality security policy. Another element that we can notice in the simple example presented is the retract command presented in both of the rules. This command is used to delete an object from the logic session.

We can observe that there is another rule in our example , RuleB_Violation. This rule is fired when an IP that is not contained into the whitelist is inserted into the logic

session. Consequently, after the initial run that provided us with satisfaction alert as a monitoring result, in case **A('23.12.31.02')** it is inserted into the logic session. The first rule will be inserted into the RuleA and will successfully continue to the 'then' part. In the second phase though, the rule RuleB_Satisfaction <*condition*> part is violated so it won't surpass the 'then' part of the rule, unlike the rule RuleB_Violation <*condition*> part, that is satisfied and produces the defined alert for the violation of the security policy.

Focusing more on the RETE algorithm, we can observe how it handles the facts and depict the example's outcome as logical steps. When defining the nodes, we can present the Rete's algorithm terminology which is included in ***Figure 3.5***.

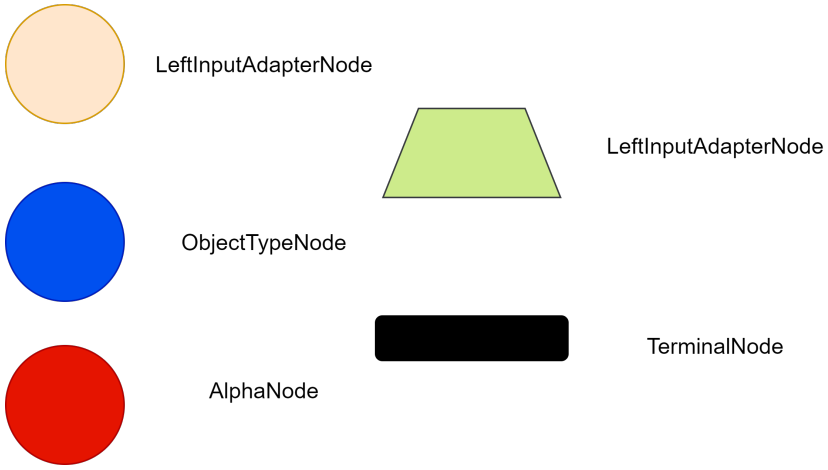

Figure 3.5: Drools node representation definitions for the RETE algorithm.

As mentioned, Drools is a node based language. We describe these nodes in ***Figure 3.5*** and explaining their role:

- ObjectTypeNodes can propagate to AlphaNodes, LeftInputAdapterNodes and BetaNodes. AlphaNodes are used to evaluate literal conditions.
- Drools extends Rete by optimizing the propagation from ObjectTypeNode to AlphaNode using hashing. Each time an AlphaNode is added to an ObjectTypeNode, it adds the literal value as a key to the HashMap with the AlphaNode as the value. When a new instance enters the ObjectType node, rather than propagating to each AlphaNode, it can instead retrieve the correct AlphaNode from the HashMap, thereby avoiding unnecessary literal checks.
- To enter the network, we use a LeftInputNodeAdapter - this takes an Object as an input and propagates a single Object Tuple.
- Terminal nodes are used to indicate when a single rule meets all its conditions, the rule has a full match. A rule with an 'or' conditional disjunctive connective results in sub-rule generation for each possible logically branch; thus, one rule can have multiple terminal nodes.

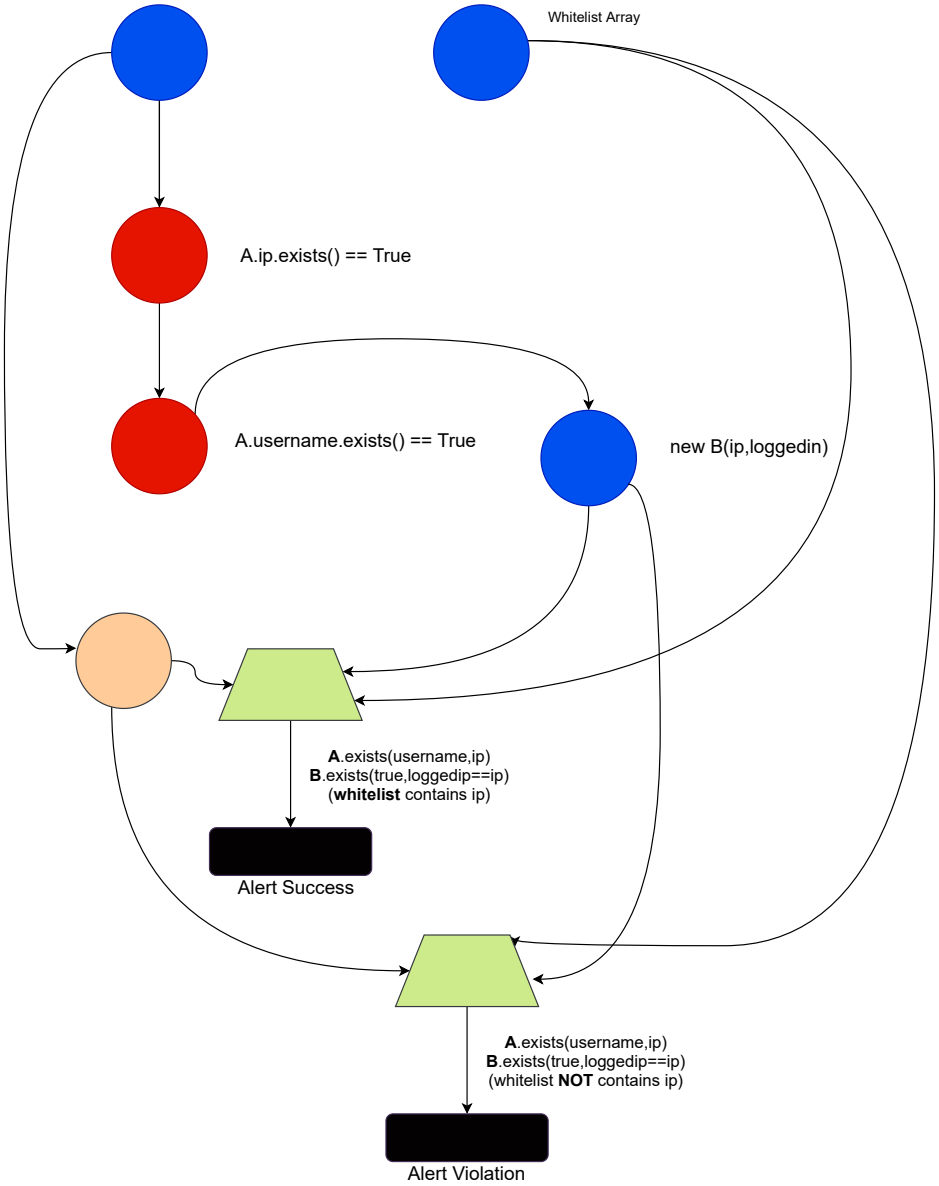

Figure 3.6: Simple Example Confidentiality rule logic represented with Drools Nodes. Definitions arise from the node declaration on ***Figure 3.5***.

This node schema description is utilized as a basis to start to building blocks together and envision the simple rule of confidentiality that we defined above. As we stated previously, this confidentiality rule uses the node sharing feature that the Drools engine provides. In ***Figure 3.6***, we examine the two Confidentiality rules that we defined which produce two types of alerts, one for Violation and one for Satisfaction. Thus, we can observe the starting node and the global value of Whitelist Array in the top tier of the nodes and the checks that evaluate an event. First, there is a check referring to the body of the

event, checking if there is an IP field. Then the second Alpha node checks if a username is contained into the event body. The assumption that is fired if the event is complying with the previous evaluation steps, is producing as an outcome another ObjectTypeNode that is containing the information of the B object (ip:String, loggenin:Boolean).

As for the conclusion of the rule, we can see that a logicalNode is used and either satisfies the predicate or violates it by taking into consideration the Whitelist array global variable and producing in either case a Terminal Node that alerts for the outcome.

"The only way to make software secure, reliable, and fast is to make it small."

Andrew S. Tanenbaum

Chapter 4

System Implementation

4.1 Technical Backround

Everest Module is a continuously evolving tool used to perform assessments in various types of organizations. This is a runtime monitoring engine built in Drools and Java that offers an API for establishing monitoring rules to be checked (e.g., set of rules defining security-related criteria or policies to be evaluated). The role of the module is to forward the runtime events from application's monitored properties and finally obtain the monitoring results as mentioned above.

4.1.1 Java

For choosing the right programming language, we take into consideration above all the security; a memory safe language was an one way decision. Also, we had to consider that the module of Drools, to which the logic of our tool is depending on, is well defined and regularly maintained in a Java implementation. Java is a high-level, class-based, object-oriented programming language designed to have minimal implementation dependencies. It is executed by the Java Virtual Machine (JVM), as explained in Venner's work [33]. The newest version that Java provides is the Java 11. Oracle released Java 11 in September 2018, only 6 months after its predecessor, version 10. Java 11 is the first long-term support (LTS) release since Java 8. Oracle stopped supporting Java 8 in January 2019. As a consequence, many development activities have evolved from older versions to acquire the newest features. This makes it easier for developers to get access to performance, stability, and security updates.

In terms of features, Java 11 added 17 new Java Enhancement Proposals. The new features that were crucial to prefer Java for implementation language of our tool, besides the Drools Java stable implementation, where characteristics that revolve around boosting runtime and security establishing between services involved, are as follows:

- Nest-based access control
- Dynamic class-file constraints

- Epsilon: A No-Op Garbage Collector
- HTTP Client (Standard)
- Local variable syntax for lambda parameters
- Unicode 10 support
- ChaCha20 and Poly1305 cryptographic algorithms
- Launch single-file source-code programs
- Low-overhead heap profiling
- TLS 1.3 support
- ZGC: A scalable low-latency garbage collector

It also maintained in a six-month period, making the Java language an ideal choice for implementing a tool that delivers security services. Additionally, Java has significant advantages over other languages and environments that make it suitable for just about any programming task. The advantages of Java are as follows:

- Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.
- Java is object-oriented.
- Allows to create modular programs and reusable code.
- Java is platform-independent.

One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Because of Java's robustness, ease of use, cross-platform capabilities and security features, it has become a language of choice for providing worldwide Internet solutions (AIX, 2020). Also, the memory safety feature of the language and the monitoring tools that it provides, it allowed us to create a robust stable and easy to understand codebase, considering also that Drools creates and uses an In-memory Database which itself demands a focus on the RAM consumption.

4.1.2 Framework

The main consideration of the EVEREST is to become a very light weight service and provide functionalities in an understandable and scalable manner. The aim is to integrate EVEREST into lightweight base systems, such as Raspberry Pi and other IoT devices, natively, or in a distributed cloud environment. Spring Framework offers a dependency injection feature that enables objects to define their own dependencies, that the Spring container later injects into them. This enables developers to create modular applications consisting of loosely coupled components that are ideal for microservices and distributed network applications.

Spring Framework also offers built-in support for tasks that are typical for applications to perform, such as data binding, type conversion, validation, exception handling, resource and event management, internationalization and more. It integrates with various Java EE technologies, including RMI (Remote Method Invocation), AMQP (Advanced Message Queuing Protocol), Java Web Service, and others. In sum, Spring Framework provides developers with all the tools and features the need to create loosely coupled, cross-platform Java EE applications that run in any environment. Java Spring Boot (Spring Boot) is a tool that simplifies the development of web applications and microservices making them faster and easier to deploy through three core capabilities:

- Autoconfiguration.
- An opinionated approach to configuration.
- The ability to create standalone applications.

In more detail:

- **Autoconfiguration** stands for applications that are initialized with pre-set dependencies that you don't have to configure manually. As Java Spring Boot comes with built-in autoconfiguration capabilities, it automatically configures both the underlying Spring Framework and third-party packages based on your settings (and based on best practices, avoid errors).
- Spring Boot uses an **opinionated approach** to adding and configuring starter dependencies, based on the project's needs. Following its own judgment, Spring Boot chooses which packages to install and which default values to use, rather than requiring you to make all those decisions yourself and set up everything manually. This is a feature that is very helpful for fast integration of a component.
- Spring Boot lets you create **standalone applications** that run on their own, without relying on an external web server, by embedding a web server such as Tomcat into your app during the initialization process.

Also, SpringBoot offers a wide range of embedded security features and easily integrated Unit testing and integration testing features which is a great add-on for the production level pipelines of a system. A core structural component of the application itself

that we are building, is the Rest API layer that is used to initiate and handle assessment executions. Springboot is structured in a way that allows easy integration of such kinds of technical requirements.

4.1.3 Maven

Maven is a popular open-source build tool developed by the Apache Group to build, publish, and deploy several projects simultaneously for better project management. The tool allows developers to build and document the lifecycle framework. Based on the Project Object Model (POM), this tool has made the lives of Java developers easier by enabling them to develop reports, check builds, and set up testing automation. Maven focuses on simplifying and standardizing the building process by handling the following tasks:

- **Builds**: Wrapping up applications and provide a structured building.
- **Documentation and Reports**: There are open-source tools for getting a number of reports and metrics for a project.
- **Dependencies**: Promotes modular design of code. by making it simple to manage multiple projects it allows the design to be laid out into multiple logical parts, weaving these parts together through the use of dependency tracking in pom files.
- **Distribution**: Maven reduces the size of source distributions because jars can be pulled from a central location.
- **Releases**: There is a regular maintenance in the maven community.

Maven supports dependency management and will retrieve them transitively and gives the tooling we need to manage the complexity inherent to dependency management: There is the option of analyzing a dependency tree and of controlling the versions used in transitive dependencies. It is also possible to exclude some of them if required, control the converge across modules, etc. In parallel, Maven helps on wrapping the EVEREST application by providing readability and easy to integrate DevOps solutions.

4.2 Drools Axioms Integration

In order to have those axioms defined, we had to -at first- define a static rule-set that is loaded into the knowledge base, before any given rule. Subsequently, we defined 3 plus 1 Drools rules which are implementing the Event Captor Axioms functionality.

Axioms in Drools, ***Table 4.1***, are serving all the purposes that the axioms on Event Calculus serve for the logical perspective of the language. We define them with the salience 20 to have absolute priority whenever a new event is about to be processed. In more detail:

- **HappensInitiates** Axiom rule is serving the purpose of ensuring that no Initiates for the same Fluent can be initiated at the same time and makes sure that when concurrent events are fed in the knowledge session, there will be some priority withheld if both events are initiating Fluents.
- **Terminates** Axiom is serving the purpose of retracting the Fluent from the Knowledge base.
- **Initiates** Axiom is serving the purpose of inserting the Fluent from the Knowledge base.

Table 4.1: EC Axioms on Drools Syntax

```
Axiom 1:
rule "HappensInitiates" salience 20
   when
      he : Happens (me: e , mt1: t)
      f1 : Fluent (name == "f1" )
      not Initiates(e == me , f == f1 , t == mt1)
   then
      insert(new Initiates(e1, f1, mt1) )
end

Axiom 2:
Rule “Terminates" salience 20
   when
      termin : Terminates(me: e, mf: f, mt1: t)
      f1: Fluent(ID == mf.ID)
      HoldsAt(f == mf , t == mt1)
   then
      retract(f1)
end

Axiom 3:
Rule “Initiates" salience 20
   when
      Initiates(me: e, mf: f, mt1: t)
      not HoldsAt(f == mf , t == mt1)
   then
      insert( new Fluent(mf, mt1) )
end
```

We should keep in mind that we must be very careful with the Knowledge base and its status, given the fact that it is implemented by the Drools into an In-Memory Database.

Every 'junk' that we might not 'clean' can have implications, so we must be very careful with the insertions and retractions of objects (Happens, HoldsAt, Initiates, Terminates). Now, we have to also define an extra Axiom in order to keep the memory of the system stable and sustainable.

Table 4.2: Retract Events from Memory

```
rule "Retracting_Events" salience 30
   when
      h1: Happens (_e1: e, t1: t)
      depr: Deprecated(happens == h1)
   then
      retract(depr)
      retract(h1)
end
```

Retracting_events rule ***Table 4.2*** is responsible for keeping the in-Memory database clear from Deprecated events. We Define the Deprecated Object as a tab and every time the main rule is resolving a Satisfaction or violation, the events (Happens) that were used are tagged as Deprecated and the Retracting events removes them from the Knowledge base freeing up the in-memory Database and keeps our run-time application lightweight.

4.3 EVEREST Example Rule

Extending the chapter in which we defined the axioms earlier, we are incorporating a representation of an already implemented EVEREST rule that is provided as a state of reference for the distinctive simplicity that it showcases, as well as the on-the-spot impact for its security importance in a cyber system.

The rule that is explained in this chapter is an extended example of the simplest rule that is written into Drools syntax. This rule checks the availability property of an asset and is depicted in 3 different aspects: The ECAssertion representation of the rule that is very well compacted, the strict Event Calculus representation, and then the Drools representation that we produced and made the necessary conventions, in order to map with the EC representation.

In particular, this EC rule checks that availability of a service, by requiring service replies within specific timeframe; in this instance the timeframe is 10 milliseconds. This timeframe can be extended at will, depending on the importance of an asset that will be assessed. Per example, a printer that is a typical noncritical asset of a CyberSystem may be assessed in terms of 10 seconds unavailability but a high-performance database that runs into a server or VM and contains PII information should not be unavailable and even if a second of unavailability occurs, this incident has to be reported immediately.

Table 4.3: ECAssertion Availability rule representation

FORALL _e1, type, _callID, _t1, _arg1, _arg2:
Happens(e(_e1, call (_arg1, _arg2), _t1[_t1, _t1])
EXIST _e2, _t2:
Happens(e(_e1, call (_arg1, _arg2), _t1[_t1, _t1])

Table 4.4: Availability Violation in EC

Violation Rule
FORALL _e1, type, _callID, _t1, _arg1, _arg2:
Happens$(e(\_e1, call(\_arg1, \_arg2), \_t1[\_t1, \_t1])$ **and** ¬**Happens**$(e(\_e2, res, (\_arg1, \_arg2), \_t1[\_t1, \_t1 + 10])$ **and** ¬**Initiates**$(\_e2, Violation, \_t2)$ $\Rightarrow$ **Terminates**$(\_e1, Success, \_t2)$ **and** ¬**Initiates**$(\_e1, Violation, \_t2)$

Table 4.5: Availability Satisfaction in EC

Satisfaction Rule
FORALL _e1, type, _callID, _t1, _arg1, _arg2:
Happens$(e(\_e1, call(\_arg1, \_arg2), \_t1[\_t1, \_t1])$ **and Happens**$(e(\_e2, res, (\_arg1, \_arg2), \_t1[\_t1, \_t1 + 10])$ **and** ¬**Initiates**$(\_e2, Satisfaction, \_t2)$ $\Rightarrow$ **Initiates**$(\_e2, Satisfaction, \_t2)$

As we can observe, the first representation of Event Calculus Availability as represented in ***Table 4.3***, two rules, ***Table 4.5*** and ***Table 4.4*** are the Theoretical ones that combust the logic into two simple drools syntactic rules ***Table 4.6*** and ***Table 4.7***. These rules have the same salience (0) and events are evaluated in the same layer for each rule. In our case it is clear that the representations of Drools syntax and Event Calculus are very close, adding some constraints it is visible that we can automate the procedure of parsing Event Calculus and create embedded EC on Drools easily.

Table 4.6: Availability Satisfaction in Drools

```
rule “Rule1_Success"
   when
      Happens (_e1: e, "call" == e.type, _arg1: e.args[0], _arg2: e.args[1], t1 : t)
      Happens (_e1: e, "call" == e.type, _arg1: e.args[0], _arg2: e.args[1], t2: t,
      TimeBounds(t2, “[”,t1, t1+10,”]”))
      f1: Fluent(name == "SuccessfullFluent_R1")
      not Initiates (e == _e2, f == f1, t == t2)
   then
      System.out.println("Rule 1_Success ");
      Insert (new Initiates (_e2, f1, t2));
end
```

Table 4.7: Availability Violation in Drools

```
rule “Rule1_Violation"
   when
      Happens (_e1: e, "call" == e.type, _arg1: e.args[0], _arg2: e.args[1], t1: t)
      not Happens (_e1: e, "call" == e.type, _arg1: e.args[0], _arg2: e.args[1], t2 : t,
      TimeBounds(t2, “[”,t1, t1+10,”]”))
      f1: Fluent(name == "ViolationFluent_R1")
      not Initiates (e == _e2, f == f1, t == t2)
   then
      System.out.println("Rule 1_Violation ");
      Insert (new Initiates (_e1, f2, t2));
end
```

From the program language’s perspective, we have created the Happens() class that handles all the events that are fed into the logic session. Happens contains event objects and timestamps them due to the fact that time resolution is a must-have input for evaluation for the most of the rules’ assessments. Apart from the “must" attributes that an event contains, Happens() class contains a function for evaluating time in a mathematical manner (Timebounds() function). This function can handle the received predicates as a mathematical formula and deduce outcome for the inserted timeline values returning a Boolean variable.

Taking now into consideration the next line of the rule, we are specifying the Fluent that this rule runs. In more detail, before triggering a rule, the monitoring core is aware of the rule that it is going to run. Thus, it predefines for every rule two fluents and inserts them into the logic session. Later, these fluents are used in order to enable the rules that are running in the knowledge session and there is a capability of adding on runtime rules to be checked, although this occasion is rather rare.

Moreover, we added the **not Initiates** part which is used in order to achieve a serial storage of the results in case for example two violations are produced in the same timepoint. These variables hold the resulting of the rule until one initiate is available.

In Drools we can also write native Java commands into the **then** part so for the visibility we are printing the outcome of the evaluation and afterwards we update the Violation or Satisfaction fluents in order to sync them with the new timestamps and the new event that either satisfied the rule or violated it.

In summary, we observed how the integration of the rules is implemented and the syntax that we defined is very close to native Event Calculus solutions. Furthermore, we have created classes that match the EC syntax and that evaluate multiple rules with high efficiency rate.

4.4 Event and Result Structure

The raw data of the sensed system variables (i.e., with Beats and Event Captors) are stored in the Elasticsearch Database, forming the Knowledge Base of the solution. This also comprises the raw evidence, based on which the overall security- and privacy-related results are disclosed. Therefore, the components that need to exchange information with the Monitor have to read/write messages via Broker queues. The content of the messages is a formatted JSON string. Each message represents one EC event. The JSON format and other technical information are described in detail in the following sections. In short, an event contains information regarding:

- *_id* is a unique identifier of the event,
- *_sender* is the identifier of the system component that sends the message/operation call/response,
- *_receiver* is the identifier of the system component that receives the message/operation call/response,
- *_type* is the processing status of an event (i.e., REQ if the event represents an operation invocation and RES if the event represents an operation response),
- *_arguments* is the signature of the dispatched message or the operation invocation/response that is represented by the event, comprising the operation name, operation instance and its arguments/result,
- *_source* is the identifier of the component where the event was captured.
- *_time* is the specific timestamp that the event was triggered into the system.

Providing a more specific view, the event structure is introduced and follows a JSON representation as in the following format:

Listing 1 Event Structure JSON Representation

```
{
"Id":(long), //unique id for event
"type":(String), //row defines if the event is a call or a response event [type of the event]
"sender":(String), //Depending on the type. Asset name or Event Captor
"receiver":(String), // As sender ; Event Captor or Asset name
"arguments": ["_Opname","_opInst","arg1","arg2"], //Metadata
"time":(long milliseconds), //When the event happened
"source":(String) // Which beat gave the relevant raw info
}
```

The monitoring assessment results hold various monitoring-based parameters such as:

i the outcome of the monitoring process (satisfaction if a monitoring rule was satisfied and violation in any other case), and

ii the events involved in the evaluation.

The outcome is very flexible in order to provide an easy integration with other tools that may be consuming these results. The default representation of the results though that is stored to a Database and can be also produced as a JSON format and be forwarded to other applications that are doing an extended analysis of these results is:

Listing 2 Assessment Result JSON Representation

```
{
"assessmentresultid":(Long),
"assesssmentprofileid":(String), //Type of the assessment execution that is checked
"assessmentCriterionid":(String), //Specific profile that brought the outcome of the assessment
"criterionDescription":(String), //Explain the rule that is responsible for this outcome
"assetid":(Long), //Explain the rule that is responsible for this outcome
"securityproperty":(String), //Which asset produced this result
"result":(Boolean), //Under which high level concept this profile is referring to (CIA)
"ifevent":(JSON), //Which is the event that is in the left hand side of the rule
"thenevent":(JSON), //Which is the event that is in the right hand side of the rule
"timestamp":(long milliseconds), //When the event happened
"variable":(String) //Which beat gave the relevant raw info
}
```

IfEvent and **ThenEvent** fields on the JSON structure have the structure of the Event Structure format that was abovementioned.

4.5 External Components

The EVEREST consumes and publishes data via a Message Broker (RabbitMQ or Kafka). When the Security and Privacy Assurance Platform (SPAP) initiates a new assessment for a set of security or privacy properties, it establishes a specific queue in the Broker in order to exchange information with the monitored system. A security/privacy assessment by the Assurance Platform is supporting the following interactions:

- **GET** data from:

  - External sources concerning the information and knowledge that are transferred from the local systems to the backend. These are mainly including recorded events and identified incidents.
  - Beats, that are lightweight log shippers integrated with the Elasticsearch stack (will be extended later), are controlled by the Assurance Platform regarding events and incidents that are directly assessed. This involves monitored security criterions for the Confidentiality, Availability, and Integrity (CIA) of system elements that are visible by the backend infrastructure (e.g. the availability of healthcare services), as well as user privacy criteria.
  - Customised Event Captors controlled by the Assurance Platform that cover cases that cannot be monitored by the pre-defined functionality of Beats.
  - The internal AutoML module, which assesses and classifies ongoing events based on historic data and notifies the Monitor accordingly.
  - The Unified Dashboard concerning potential user actions has to be forwarded and executed by the local Primary Agents.

- **SEND** data to:

  - The Database concerning high-level information and major events/incidents
  - Message Brokers in order to serve results in a predefined queue and let any other analytics service use them. For example, any incident response tool that is correctly configured to collect information about the assessment results can analyze them and perform a set of pre-defined actions, such as executing a script to change firewall or system configurations.
  - AutoML to request to assess on-going events and classify wherever they constitute normal behaviour, anomalies, or identified malicious activities.
  - Beats and customized Event Captors in to start, pause/resume, or stop their operation.

So in this chapter we analyse all the components that are combined with the EVEREST to provide the necessary modelling, view, and additional integration capabilities to our solution.

4.5.1 Asset Loader

Asset Loader Module is essential for the functionality of the security assessment platform. It contains the definitions of the organizations, assets, projects inside the organizations as well as assessment criteria (rules and assumptions that are used) and assessment profiles. The component is responsible for receiving the security assurance model for the target organization. This model includes the assets of the organization, the security properties for these assets, the threats that may violate these properties and the security controls that protect the assets. The Asset Loader component is based on an assurance model that encapsulates definitions that holistically describe organizations, assets, and assessments.

4.5.2 Unified Dashboard

Assurance GUI is deployed in Angular and is the integrated version of the Assurance Platform that provides the visual aspects of the asset loader and supports the solutions that are integrated.

4.5.3 PostgresSQL Database

This Relational Database Management System (RDBMS) used to contain all the information the asset-loader uses, as well as many extra fields that contain assessment results and metrics from all the security assessment tools regarding the assessments.As a complete solution, this database depicts the Assurance Model Solution that is referenced in the Asset Loader Description and aggregates all the information needed to define assets and aggregators for assessment results from all the tools used to define the security posture of an organization (Vulnerabilities loader, Penetration Tester, and EVEREST).

4.5.4 Event Captors

An Event captor's generator is a tool that, based on the collected data and triggering events, formulates a rule or a set of rules and pushes the latter towards monitor module for evaluation. Data and events are mostly collected through Elastisearch based on lightweight shippers (namely Beats) such as Filebeat, MetricBeat, PacketBeat etc. which forward and centralize log data. Data can also be collected through Logstash, an open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to ElasticSearch. The process of aggregating information through Beats is extensively analyzed in later chapters. The Event captor's generators tool is initiated through REST calls from the monitor module. This tool is essential for the EVEREST functionality due to the fact that it collects and feeds essential information from assets as events.

A more detailed view of the Event Captor will be provided in the next chapter, explaining the information that is collected, mapping the Beats in security policy categories based on the gathered information, and the pipeline that produces the final monitoring structured events.

4.5.5 Message Broker

Rabbitmq is responsible for carrying out the message bus that feed the EVEREST with events using Message Queuing. Message queuing allows web servers to respond to requests quickly instead of being forced to perform resource-heavy procedures on the spot that may delay response time. Message queuing is also useful when we want to balance loads between workers and distribute messages in a reliable manner. message broker using dedicated message queues can also help create solutions that handle execution errors.

For example, if the consumer service is terminated due to any error Queue is holding the bunches of elements that are received (events in our case) and when the service is up and running again, those elements are consumed and reasoned without completely

eradicate the functionality process of the tool. RabbitMQ in particular is a good fit for our case due to its lightweight, reliable integration, and overall performance. These factors make our tool more flexible for scaling up.

4.5.6 AutoML

The AutoML module is based on the Auto-Keras framework [34] and can process the already gathered unified information in a similar fashion as the Monitor module. Therefore, the AutoML module, can access the Knowledge Base (i.e., Elasticsearch) and use the data to create training and evaluation datasets for supervised learning. The training is performed on past data. Internally, the AutoML module assesses the success rate of various ML algorithms and selects the best solution. A ML model is built based on this training dataset the best performing algorithm. When the Monitor requests the evaluation of current events, the AutoML utilizes this ML model to classify them and send back the result. This is an internal communication within the Assurance Platform via a specified queue in the Broker (similarly as Beats, Event Captors, and Metadon exchange information with the Monitor).

4.5.7 Incident Response

Incident Response (IR) tool is a tool developed to holistically preview and mitigate a possible cyberattack. IR is an organized approach to address and manage the aftermath of a security breach or cyberattack. The entire process is documented with files, usually free text, called “playbooks”, which can be shared across organizations. Playbooks provide graphs containing nodes that roughly map to executable actions. IR playbooks provided by security agencies and organizations are not executable workflows and these solutions provide playbooks that are not for free. Opensource or interoperable between tools. We developed a generalized tool in which we can create these playbooks and have a structured plan for an instance of cyberattack. This tool works standalone or in collaboration with EVEREST. By noticing violations in security properties that EVEREST produces, EVEREST, when a violation of a rule is triggered, can indicate a playbook that should be initiated. Also, vice versa IR can indicate that a cyberattack is prevented on a specific asset and trigger a monitoring assessment for this asset to observe the overall wellness of the asset after the cyberattack.

“Computer Science is no more about computers than astronomy is about telescopes.“

E. W. Dijkstra

Chapter 5

System Architecture

In this section, we will present the Architecture of the system, after analysing some key aspects of the integration and how we have defined the initiation and termination process. First, we present, how EVEREST initiates an assessment, what are the primary information producer entities and how they are integrated into the assessed system. Then, we will investigate the reasons for using 2 different types of deployment architecture, extend the knowledge on both of the versions and view the UML representation, as well as holistic architecture of EVEREST and the external components deployed in various environments.

5.1 EVEREST components

The EVEREST as a tool is composed by mainly 2 components, the monitoring Orchestrator and the monitor instance. The **monitoring orchestrator**, which is a privilege docker-container (meaning that it has the capability to create containers within itself) is responsible to communicate with the event captors to initiate the event collection from the Elasticsearch. Simultaneously, it produces another docker container, namely **monitor instance**, that is be solely responsible for handling a single monitoring assessment execution and contains all the Drools logic that we analysed in previous chapters.

5.2 EVEREST Initiation

The initiation of the EVEREST monitoring assessment tool, can be made: (a) automatically by utilizing SPAP's GUI or (b) manually, by utilizing the REST APIs provided by EVEREST. The expected input is provided below. It is sent as a POST REST API request into the monitoring orchestrator module. A number of the expected fields (e.g., assessmentProfileId, organisationId etc.) are retrieved through the Assurance DB, thus they must pre-exist there.

Listing 3 Initiation POST message

```
{
"assessmentprofileID":(Long), // Assessment type (CIA)
"creatorID":(Long), // Who created the assessment
"organisationID":(Long), //Organisation that triggered execution
"projectID":(Long), // Project containing assessed asset
"assetID":(Long), // Asset that is being assessed
"beggining":(Timestamp), // Execution Initiation timestamp
"ending":(Timestamp) // Execution Termination timestamp
}
```

5.3 Event Captor components

The Event Captor orchestrator is responsible of initiating a connection with existing Beats (part of the ELK stack) that are integrated into the assets to deliver information from logs or other sources to the Elasticsearch. Some Beats and their specific information is shown in ***Table 5.1***.

Table 5.1: Beat operation & Data mapping

Operation	Beat Name
Audit Data	Auditbeat
Log Files	Filebeat
Cloud Data	Functionbeat
Availability	Heartbeat
Metrics	Metricbeat
Network traffic	Packetbeat
Windows event logs	Winlogbeat

Beats can send data directly to Elasticsearch or via Logstash, where data can be further processed and enhanced, before visualizing it in Kibana. Every beat contains a number of modules that indicate the locations of the logfiles. By enabling these modules Beats can narrow the information that will be aggregated and pinpoint the significant logs that will perform assessments.

Currently, SPAP is using the following beat systems: Filebeat, Auditbeat, Metricbeat, Packetbeat, Winlogbeat.

Every Beat can be mapped with one or more security assessment criteria, and one or more criteria can relate with one or more beats. This mapping is a process of context of every rule that is depicted into the assessment criteria and the responsible logs that gather the specific information.

5.3.1 Beat installation example

To give an example, for the rule of the confidentiality of the logins that we demonstrated earlier, the responsible beat that we have to deploy in order to get the login triggers from an http server, is the Filebeat [35]. To bring the terminology into the Assurance layer, we

define a hardware asset, such as a specific http server, and install the Filebeat within this asset. The Filebeat is responsible for acquiring log information and providing it as raw events to EVEREST.

Filebeat is a lightweight shipper for forwarding and centralizing log data. Installed as an agent on the chosen server (in our case the http server), Filebeat monitors the log files or locations that we specify, collects log events, and forwards them to Elasticsearch for indexing. Filebeat of course is a huge beat aggregating many logfiles and presenting them into the Elastic Database as entries that have multiple fields. For this reason, we must indicate to the Beat which fields are important for our case. In our case we indicate the Filepath for the logfile and the information, and we enable the system and the auth log in order to aggregate the logins of the server. These modules are connected to the /var/log/auth.log and the /var/log/system.log of the http server respectively and gather information that push into the ELK database.

The Event Captor module is now triggered by EVEREST to read all these entries from the ELK database (or from a specific point in time and afterwards). It modifies the entries in the specific event-like form mentioned above that EVEREST understands and feeds all those events to EVEREST.

In conclusion, every time any user logs into the http server, the information is gathered and fed into EVEREST, and it is also stored into the ELK database to be used for historic data from other sources.

5.3.2 Communications

The communication between the event captors and the Beats is end-to-end encrypted. As an extra step, the Event Captor Orchestrator gathers information from the Elasticsearch and parses it to fit the predefined event structure. This process is being performed to ensure the eligibility and validity for the reasoning process that is taking place into the Monitoring instance. Monitoring instance process will be explained furthermore in an upcoming section.

The Event Captor orchestrator then subscribes to a predefined queue and sends the events into the monitoring instance. This communication is also fully encrypted using the capabilities of RabbitMQ encryption with the Transport Layer Security version 1.3 (TLS v1.3) [36] protocol enabled.

When EVEREST receives an event from an event captor, it initiates the reasoning process and stores the output (assessment result) in a Postgres DB [37].

5.4 Architecture Versions

The main focus of the EVEREST implementation, apart from the core elements that we unravelled earlier, is that it must be as lightweight as possible. This raises the need to implement our application in a flexible, efficient, and distributed manner. "We must also keep scalability in mind while considering that our tool must be capable to serve the needs of groups of organisations with a single integration.

EVEREST tool is currently deployed in many EU projects and also is production-ready to be deployed in custom organisation systems, in order to ensure the security policies are followed in any aspect of a system. The key concept is that EVEREST is a very abstract tool that encapsulates many security policies and can be deployed and run anywhere. Although the types of the business organisations that EVEREST is deployed on varies, we must consider that the sector of interest of these organisations may differ in many ways. For example there is a big difference on a automobile organisation than a hospital organisation. The biggest differences are the amount of personal Identifiable information that each organisation contains in its premises and the deployment policies for the tools that are integrated in the organisation technical infrastructure itself.

Many organisations that we have collaborated with so far, which are directly involved with patients or very high confidential information, prefer the deployment of the EVEREST into their premises in an isolated environment in order to ensure the overall security and the confidentiality of the information that will be assessed even if EVEREST ensures end-to-end encryption for the information that is managing. On the contrary, other types of businesses that have not invested into security and want to assess the current state of their system in order to perform the projected security guidelines and to have a mature view of the security posture of their system , prefer a decentralised deployment.

Driven by and in an effort to align with the above-mentioned requirements that different types of organisations set, we produced two main versions that are used deliberately for each specific case. The dockerised version is an on-premises integration and is consider better for isolated infrastructures and the other version includes a Cloud based integration that is deployed on Kubernetes cluster and has more flexible and performance lighting characteristics. To recap, the versions that are provided, can be distinguished from their main characteristics referred as:

- The **Kubernetes version**, that serves the purpose to provide to personalised applications the cloud integration (non-pii or consensus) and
- **Dockerised version** to be hosted directly in organisations that are disclosed

We will provide architectural graphs on both of versions that are implemented for EVEREST.

5.4.1 Dockerised Version

Focusing on the Dockerised version that is mainly deployed on premises of the organizations, we unravel the components that, if combined, can bring up the full functionality of the monitoring assessment tool. We present ***Figure 5.1*** EVEREST's abstract architectural view for the Dockerised integration version:

We observe in this architectural graph that the Dockerised version comprised by the following modules:

- Assurance GUI Module, or Actor responsible to trigger the initiation of assessments
- Incident Response Module responsible to trigger assessments

- AutoML Module responsible to partially interconnect with EVEREST assessments
- Event Captor Module, that contains the following architectural features:
  - Ecorchestrator
  - Elasticsearch Database
  - Beats
- Message Broker. Rabbitmq module that handles the information flow,
- Relational Database storing assessment results and state
- EVEREST module containing the following architectural features:
  - Orchestrator
  - Monitor Instances

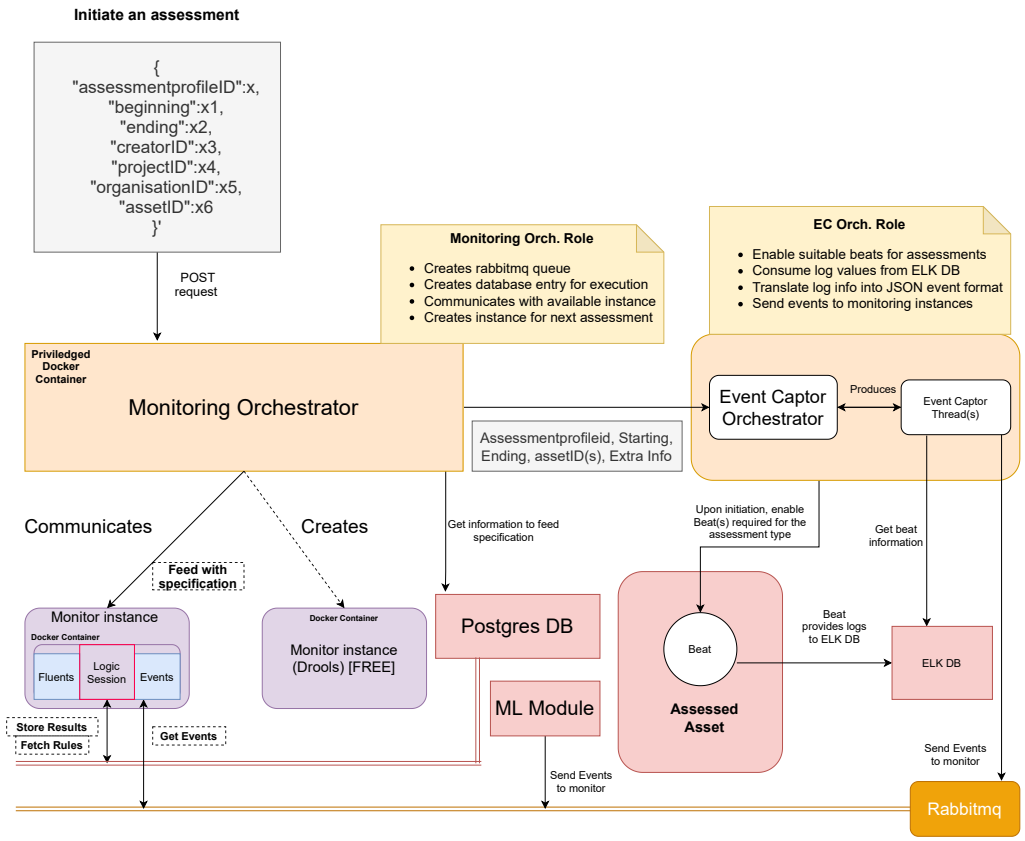

Figure 5.1: Depiction of Dockerised version architecture and flow followed in case of assessment execution. Used for the in-premises deployments on closed environment organisations.

As a functional description of the system, the flow of the information starts by triggering an assessment, either by a predefined actor (External Tool) or by the traditional way of the Assurance GUI.". We have to highlight at this point that the Orchestrator, Monitor instances, Rabbitmq ,PostgreSQL and Ecorchestrator containers are included in a custom docker network for security reasons and for easy interconnections between each other. In continuity with the functionality, the triggering mechanism, is sending a the JSON to a REST API manager, which initiates the assessment.After performing the processes required for setting up the Rabbitmq Queue for the instance, Orchestrator, starts a scheduler for the future termination of the assessment, if specified. It also spawns a new empty container for possible future assessment, transmits a renewed JSON into the monitor Instance that is created and awaits an order for assessment execution initiation. The process of the creation of an "empty" container, is performed by a sequence thread, and in the dockerized integration, the orchestrator must be a privileged docker container with docker creation rights. Thus, by using a namespace handler, orchestrator eventually will spawn a monitorassessment-ready container that will direct the future assessment request that will be made. After the request is made via sending it through the dockerised network using name resolution, the monitoring instance is performing the processes described visually into the UML, which is to:

- Define the Global Values for the assessment (Unique IDs for storing results in later stage).
- Load the Axioms into the Drools Engine from the database
- Load the satisfaction/violation Fluents of the Rule(s).
- Get the Assessment Criterions from the Postgres Database.
- Load the Assessment Criterions to the Drools Engine
- Connect to the Rabbitmq Queue created by the Orchestrator container earlier.
- Wait for consuming the events that come into the Queue
- Feed the events into the Drools knowledge base.
- Resolve and store the results into the PostgreSQL database.

A Monitoring assessment instance can also trigger a function that enables external tools, such as the AutoML module. So, if an assessment profile is executed in a monitoring instance that demands the involvement of the ML module, the Drools after some initial events that correspond with the structure of the rule, will direct results into a dedicated queue into the Message Broker. The AutoML, that has already subscribed to the specific queue, will be triggered and start acting as an Event Captor for that instance, providing evidence in a form of events.

We present ***Figure 5.2***, a detailed architectural UML representation for the functionality of the Dockerised version.

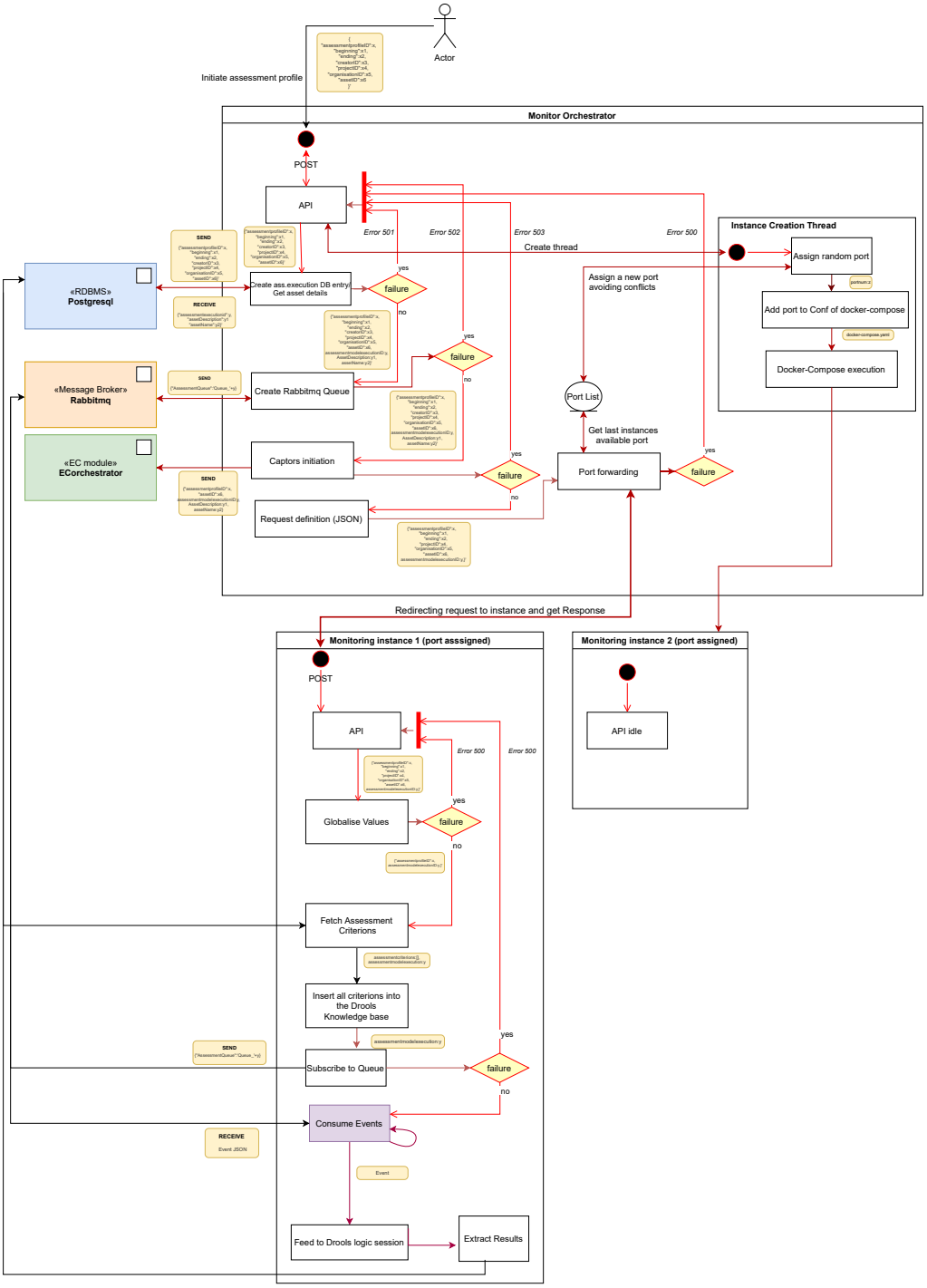

Figure 5.2: Detailed UML graph of Dockerised EVEREST Version including all the steps for an assessment execution.

Note that some integrations have been made in the Dockerised version, that allow

the deployer to extend the functionality of the instances (new features). Those features provide enhancements such as:

- Send Assessment Results to any Rabbitmq dedicated Queue
- Send Assessment Results to any Kafka integration
- Send Results directly into any ELK Database (Used for Big data analysis)

These additions were made in terms of extending implementation for different pilot deployments.

5.4.2 Kubernetes Version

We depict the Kubernetes version Architecture deployed on a group of organizations (A,B) mainly on projects that must scale up during time. This is the core The main functionalities that are described in the Dockerised version UML ***Figure 5.2***, are integrated for the Kubernetes version also. We present ***Figure 5.3*** an analytic Kubernetes architectural diagram for EVEREST.

The functionality on the orchestrator pod in this integration is slightly different to align with the cloud integrations. Mainly the production of the monitor worker container (mapping with the "monitor instance" in the Dockerised architecture), that is responsible to uphold the execution of the assessment, is made by Helm-Charts as explained in Mustafa's [38] work.

5.4.2.1 Helm

Helm is a package management tool for Kubernetes and is used to add packages, services, etc. to the APPs which are deployed in the cluster. Helm-Charts as analysed by Sherman [39], helps the user to install, spawn and upgrade any complex applications of Kubernetes. The main advantages of Helm-Charts are, as expressed by Finnegan [40], that they are easy to create, are way faster and easy to version and easy to publish and share . Helm-Charts describe even the most complex Kubernetes apps, they provide reliable application installation and serve as a single point of authority (as described by Shah at [41]).

5.4.2.2 Helm in implementation stack

In our implementation for this version, the orchestrator uses Helm Charts to instantly produce a monitoring instance (also known as a monitor worker) and spawn it as a Job.

A Job is referred to a pod that has some certain built-in functionalities. All the controllers that Kubernetes offers, share one common property: they ensure that their pods are always running. If a pod fails, the controller restarts it or reschedules it to another node [42] to make sure the application the pods is hosting keeps running. But as our tool as well, we do not want the process to run indefinitely. A monitoring assessment can have a start date and an end date. Thus Kubernetes Jobs explained extensively in [43] ensure, that one or more pods execute their commands and exit successfully. When all the pods

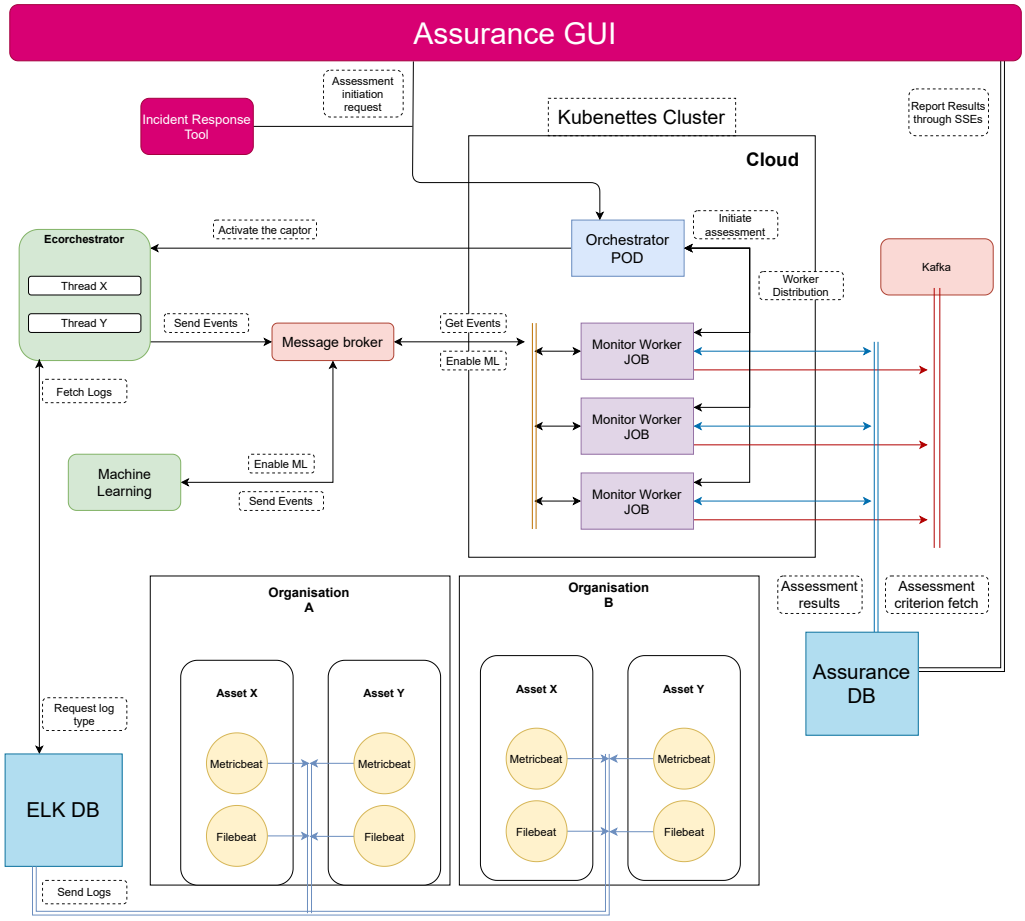

Figure 5.3: Architecture graph of Kubernetes EVEREST Version deployed on multiple organisations and flow involved on assessment execution.

have exited without errors, the Job gets completed. When the Job gets deleted, any created pods get deleted as well. In this manner we are not building an extra container that waits for future assessments as we do for the Dockerised version but we instantly configure the information in a JSON Helm Chart configuration file, execute it, and use it to spawn and load information into the worker. So now this mini-container has all the necessary information and in addition the Kubernetes manager handles by principal the execution duration.

The UML graph for this implementation is mostly identical to the one that was described for the Dockerised version. All the core functionalities are there, except the API in the monitoring instances that has been removed because the Kubernetes cluster manager can by itself control the aspects of the assessment, as well as the orchestrator does not uses the thread to produce new instance anymore. Instead of making a call it configures and executes a Helm-Chart with the information that previously was sending as a POST request to the monitoring instance.

The architecture of the Kubernetes/Helm version of the EVEREST ***Figure 5.3***, is the recommended one from our team, but we have deployed also all the components shown to the schema onto the cloud, including the Assurance GUI, DB ELK and event captors. We also have tested these architectures with multiple external tools that were integrated to communicate and interconnect with our solution.

Wrapping the above-mentioned, we managed to conceptualize and create 2 kinds of architectures, that allow our tool to scale up by demand and allow integration in all kinds of organisations with regards their accessibility to the outside world. We presented UML architectural diagram and case specific example on Kubernetes version for two organisations that shows the deployment's capabilities.

Chapter 6

Evaluation

6.1 Methodology

For the evaluation of the Drools implementation of EVEREST, we need to create some scenarios using the tools. We will compare EVEREST with the previous implementation that used native Java and a persistent storage in a relational database (Postgres) for the rule states (predicates). As mentioned, Drools uses an in-memory database, so we can see the performance boost this provides for our implementation. Additionally, the new implementation has easy deployment and flexibility, and the Dockerised/Kubernetes deployment environment makes the integration stable and efficient due to the ease of interconnections.

To compare these two different implementations, we need to define some scenarios in event calculus that are identical in each implementation. For our new EVEREST integration, we will benchmark the Kubernetes version, although the core functionality is the same for the Dockerised version. So, we need to define a standard rule defined in Drools and in XML form (the old Event Calculus representations were in XML format). Then we need to define some sub-scenarios based on the events that will be fed into each implementation.

We define a core scenario, for the one basic rule that we already referred to in the previous chapters of the EVEREST monitoring assessment tool. We will evaluate the outcome of the Availability rule which, is a straightforward ruleand includes reasoning regarding the timeseries of events. We have already defined the Availability rule in 4.6 and 4.7. We will also define three sub-scenarios per tool to evaluate the implementations:

1. The events fed to the implementations produce only violations of the rule. This scenario is chosen because one event can trigger the first part of the rule and at the same time trigger a violation, testing mainly the violation of the Timebounds. So, we have one event per every assessment result.

2. The events that are fed to the implementations produce only satisfactions of the rule. This scenario tests the tools for consistency of the predicates in two events, as for two events needed for the satisfaction so the knowledge base and different kinds

of events are evaluated. So for this scenario, we have 2 events for every assessment result.

3. The events that are fed to the implementations have 50% probability of being a violation/satisfaction. Consistency is for the tools and their real-world functionality is tested.

6.2 Experimental Setup

The evaluations for both implementations of EVEREST (Drools and Native) are performed on a Dell Precision 3551 workstation with 32GB of DDR4 RAM, an Intel Core i7-1075 processor with a clock speed of 2.6 GHz. We will also record the RAM/CPU consumption, setting a limit of 10GB of RAM.

Both implementations use RabbitMQ as a message broker, so it is convenient for benchmarking as we will use the same event sending mechanism for both tools. Additionally, the event structure is similar, so we do not have to make alterations to the benchmark script.

6.3 Archived EVEREST

In the following graphs we measure in the Y-Axis, the overall time (in ms) that an event takes to be inserted into knowledge base, evaluated by the Availability rule and extract the result. For the X-Axis we see the number of events that were sent in a row.

We provide the benchmarks for the 3 sub-scenarios per implementation starting with the old EVEREST implementation.

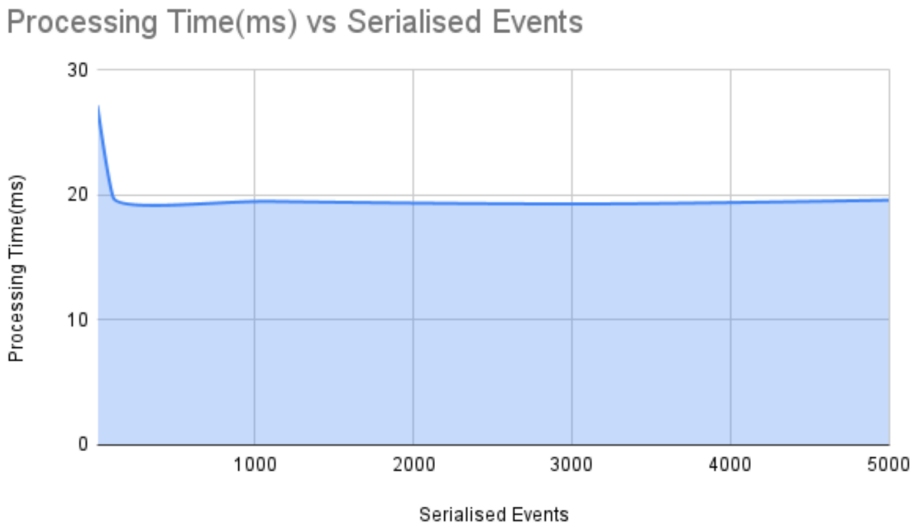

Figure 6.1: Event Processing Time of the **Native** java EVEREST availability rule, when fed with events that produce only **violations** of the rule.

We can observe that in this benchmark ***Figure 6.1***, where every event is translated to a monitoring assessment result for the availability rule, that the maximum event process time is 27.18ms for the 1st event and then after the normalization curve, it stabilizes at 19.71ms until the test is completed. Although even though the event processing time stays at stable metrics RAM consume is exceeding 10GB (threshold that we set earlier) so we conclude the benchmark processing at the 5000th event.

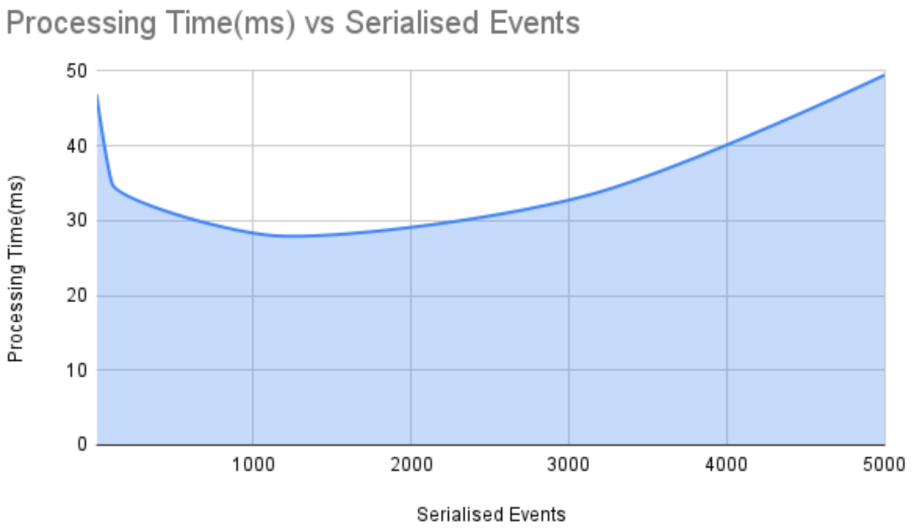

Figure 6.2: Event Processing Time of the **Native** java EVEREST availability rule, when fed with events that produce only **satisfactions** of the rule.

In the second sub-scenario, for the Archived integration of EVEREST, where every two events are translated to a monitoring assessment result for the availability rule, we can observe ***Figure 6.2*** a slightly higher evaluation time compared to the first sub-scenario. In general, we can see larger numbers for event processing times in this graph, even after the normalization curve that is present in all the graphs. However, we can see a slight linear trend for the event processing times, which remain at logical levels until the final event benchmarked. The maximum time for event processing is at 49.72ms at the 5000th event, while the minimum processing event time is 28.16ms at the 1122th event.

For the random 50/50 events that generate violations and satisfactions evenly, we can observe ***Figure 6.3*** that the normalization curve happens very quickly, with the lowest event processing time occurring at the 108th event at 26.8ms. After this point, we see a very quick adaptation to a linear model until we observe a stabilization, although the 5000th event is processed at 127.16ms.

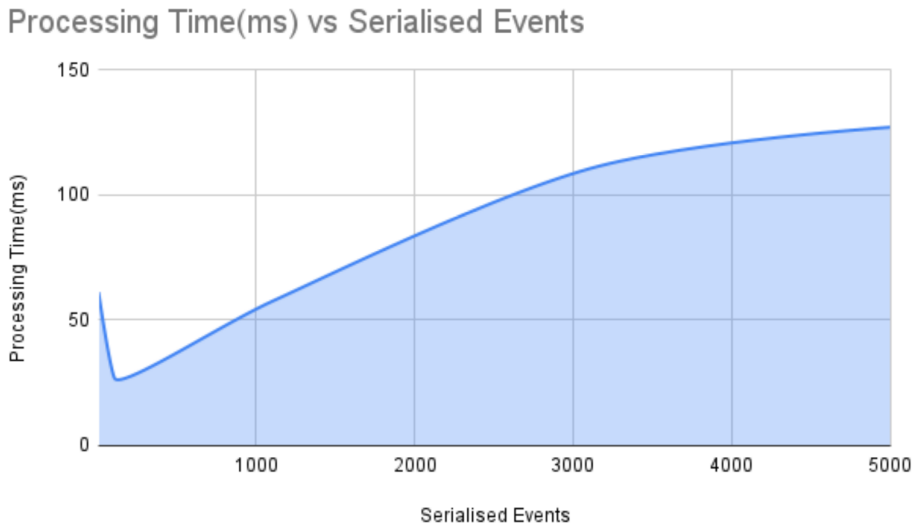

Figure 6.3: Event Processing Time of the **Native** java EVEREST implementation availability rule, when fed with events that they produce both **satisfactions and violations** of the rule.

6.4 Drools EVEREST

For the new Drools EVEREST integration, RAM consuming stayed very well below 2GB during the benchmark execution, so we decided to perform the benchmark for 20.100 events.

For the first graph ***Figure 6.4*** we observe a significant difference between the two implementations in terms of performance. Events start from 7.23ms processing time and very quickly produces a stiff normalization curve that ends at 130th event where is a drop to 0.51ms per event which stays in the same levels (0.61ms) until the end of benchmarking process. The significance of the difference in performance for the implementations comes as mentioned from the in-memory database.

For the second benchmark for the new implementation, we can see that the normalization curve that we can observe ***Figure 6.5*** in all the implementations is slightly thicker. This happens due to the fact that 2 events now, instead of the previous metric, are producing the assessment result, so drools knowledge base needs more events to adjust the processing time to 0.38ms per event.

In our last benchmark, we see the actual running of a monitoring assessment rule in the new implementation. We observe in this benchmark ***Figure 6.6*** that not only there is an almost instant performance boost, but the normalization curve that we saw in the previous graphs is almost not existent. This is due to the fact that the Drools knowledge base, very early for this use case (15th event), receives events that satisfy and violate the rule. Thus, the predicates into the logic sessions are made as a primitive state and then it standardizes the event process time at 0.41ms until the end of the benchmark.

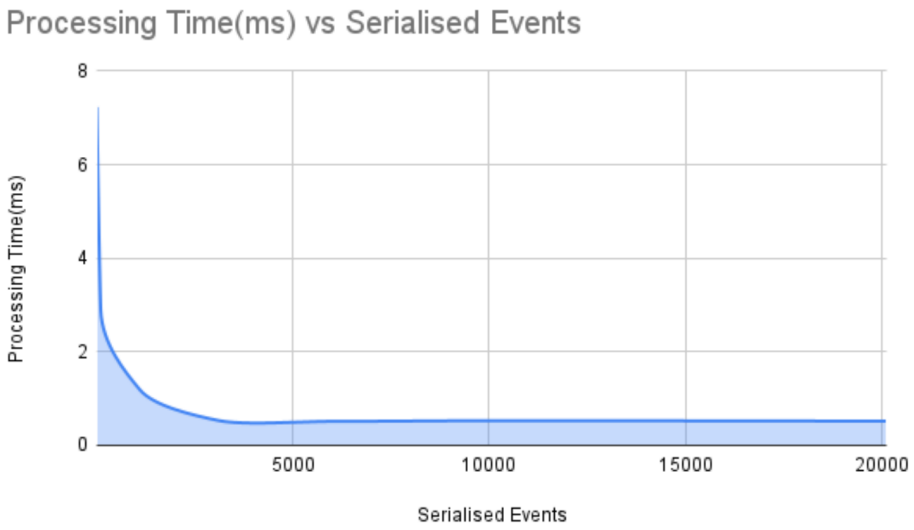

Figure 6.4: Event Processing Time of the **Drools** java EVEREST availability rule, when fed with events that produce only **violations** of the rule.

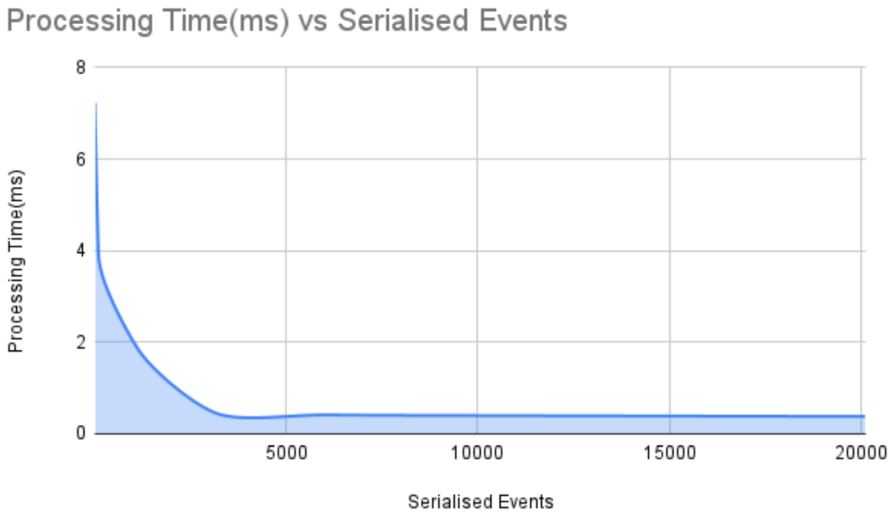

Figure 6.5: Event Processing Time of the **Drools** java EVEREST availability rule, when fed with events that produce only **satisfactions** of the rule.

6.5 Multiple parallel rules performance

We also provide a self-assessing benchmark for the Drools Java EVEREST integration. EVEREST has the capability of assessing multiple rules at the same time, giving the ability to users to check multiple security properties (CIA) in their system by running a single assessment execution. We compare in the following graph ***Figure 6.7***, the execution of the Drools EVEREST integration when initiated with multiple rules that load into the

knowledge session. In our case, we benchmark the Availability rule but with different thresholds for each rule in order to, from a perspective, differentiate the rules but keep the same event structure that is fed into the EVEREST. So, all rules are triggered, each one in different time point regarding the threshold that is automatically set (in milliseconds).

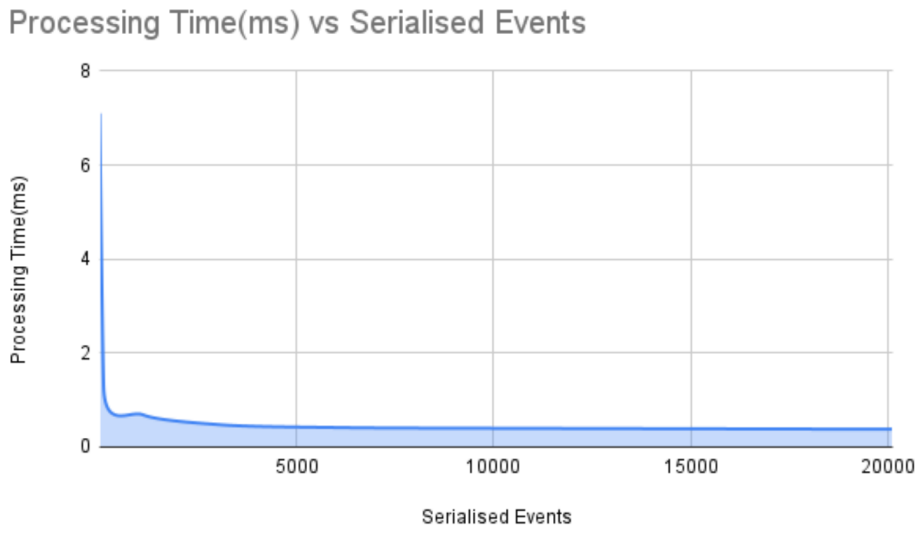

Figure 6.6: Event Processing Time of the **Drools** java EVEREST implementation availability rule, when fed with events that they produce both **satisfactions and violations** of the rule.

We run EVEREST feeding it with 20110 events 4 distinguished times. One for 1, 10 ,100 and 1000 Availability rules running concurrently, each with different threshold. We observe that the event processing time in each case quickly normalizes after the first 25-35 events and in each case the normalization stays stable until the end of the benchmark. Another pinpoint in this benchmark, is that the more the rules, the merrier the time EVEREST needs, to achieve full normalization of event processing time (PT). The only mildly significant difference appears on the 3rd and 4th occasion (of 100 and 1000 concurrent rules) indicating that the event processing time is relevant to the number of rules that are imported inside the EVEREST, but still there is nothing indication that Event PT will evolve linearly. Additionally, the cases of 100 and 1000 concurrent rules, are purposely stretched to showcase the abilities of the tool and have no real-world appliance.

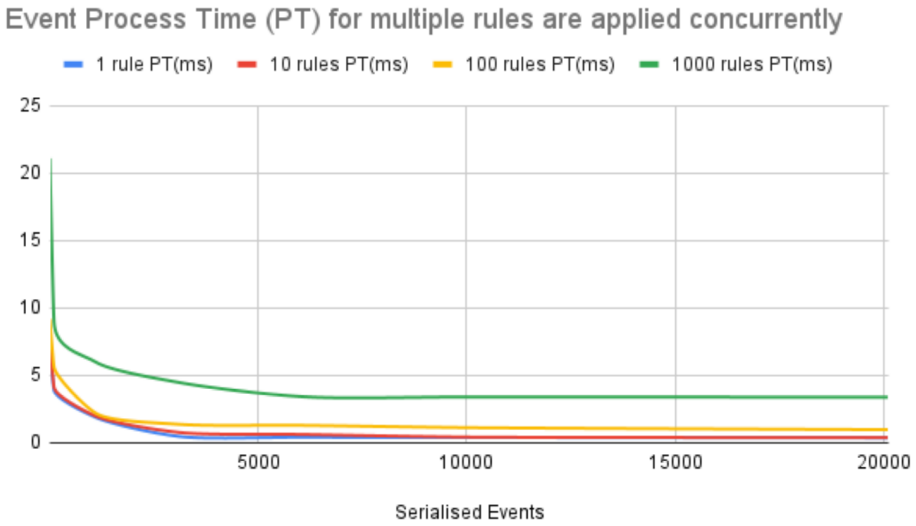

Figure 6.7: Event Processing Time of the Drools java EVEREST when multiple rules are applied concurrently fed with events that equally produce satisfaction and violation results.

"We shall do a much better programming job, provided we approach the task with a full appreciation of its tremendous difficulty, provided that we respect the intrinsic limitations of the human mind and approach the task as very humble programmers."

A. Turing

Chapter 7

Conclusion

By assessing the tools and directly compare the native Java and Drools implementation of the EVEREST, we can define with certainty that the new implementation is ideal to use in a wide scale system. Also, even by excluding the performance benchmark, the new implementation provides easy deployment, a performance steady and robust solution that can be used either for cloud environment either for in-premises integration, without influencing the logical principles of Event Calculus. The unique feature of the abstraction of the Event Calculus and having only one limitation that revolves around the capturing of the events, is kept into the new implementation, so the EVEREST can provide different kinds of assessments for a very wide spectrum of systems and services.

Overall, we managed to create a monitoring assessment tool (EVEREST) that has the following characteristics:

1. EVEREST is a system integrated with Drools in its Knowledge base core.

2. EVEREST serves the purposes of a tool capable of monitoring a complex asset wise system.

3. Event Calculus that was ported into Drools, contains abstract yet powerful rule-set that applies in all kinds of organisations.

4. Porting Event Calculus on Drools proved to provide a very reliable and stable performance wise core.

5. The multi-option deployment that was implemented, proved to be an easy-to-use and robust solution for every occasion.

6. EVEREST is capable of being deployed on cloud as well as in-premises closed environments.

7. Single deployment of EVEREST is capable of performing monitoring assessments for multiple organisations at once.

7.1 Involvement

As we mentioned, EVEREST is a tool that is appendant to the Assurance Platform which is provided as a security solution on many European Projects. Regarding this implementation that is analyzed in this research text, EVEREST has been integrated, tested and currently running onto multiple EU HORIZON 2020 Projects, namely:

1. **C4IIOT** (Cyber security 4.0: protecting the Industrial Internet Of Things [44]). In this project EVEREST is deployed on Kubernetes Cluster and holds responsibilities of performing assessments on different IoTs for malicious and anomalous behavior detection.

2. **FISHY** (A Coordinated Framework for cyber resilient Supply Chain Systems [45]). EVEREST holds responsibilities on checking high-level rules focused on trust of ICT supply chain ecosystems.

3. **AI4Healthsec** (A Dynamic and Self-Organized Artificial Swarm Intelligence Solution for Security and Privacy Threats is Healthcare IGT Infarstructures [46]). EVEREST holds responsibilities on deploying in disclosed Healthcare organisations, interconnect with other security tools as a hypervisor and run assessments that revolve around security for crucial PII.

4. **INTELLIOT** (Intelligent, distributed, human-centered and trustworthy IoT environments [47]) EVEREST responsibility is to act as a security stakeholder, performing security assessments that relate with 5g and interrelate with IoT frameworks.

5. **CYRENE** (Certifying the Security and Resilience of Supply Chain Services[48]) EVEREST responsibilities revolve around design and focus on assessments on supply chain services in sync with the new certification scheme proposed by the CYRENE project consortium.

6. **RESIST** (RESilient transport InfraSTructure[49]) EVEREST is responsible for continuous assessments performed in very specific systems and services such as Drones, that interrelate with innovation on transport infrastructure sector.

7. **SMART-BEAR** (Smart Big Data Platform to Offer Evidence-based Personalised Support for Healthy and Independent Living at Home[50]) EVEREST is responsible of assuring the security posture of this Healthcare platform that involves over 5000 participants.

8. **COLLABS** (A Comprehensive cyber-intelligence framework for resilient collaborative manufacturing Systems [51]) EVEREST provides security solutions cyber-intelligence framework for collaborative manufacturing which enables the secure data exchange across the digital supply chain. Multiple in-premises integrations of EVEREST are deployed in different infrastructure pilots.

As an addition, for the AI4HEalthsec project we structured and published a scientific paper, as a joined work with AEGIS, PDMFC and FORTH, conjoining the EVEREST implementation that aligns with the overall security solution of the project and focusing on the Incident Handling of many threats that arise into the Healthcare organizations and as a result into the supply chain ecosystem. The paper was recently accepted on the on ICTS4eHealth in conjunction with 2022 IEEE Symposium on Computers and Communications (ISCC), titled "Incident Handling for Healthcare Organizations and Supply-Chains" [52].

7.2 Future Steps

Finally, regarding future steps of this integration, as far as we have the abstraction and scalability, we work towards making the tool user friendly by adding compiler features, so that a user has the ability to directly write rules in solid Event Calculus that can be automatically translated into Drools language in the background. We are planning also to include EVEREST as a whole in a raspberry PI like device deployed, given the results of the assessments proved lightweight enough.

Bibliography

[1] B. Chess, "Improving computer security using extended static checking," 05 2002.

[2] M. Krotsiani, "Model driven certification of cloud service security based on continuous monitoring," Ph.D. dissertation, City University London, 2016.

[3] O. A. Michalec, D. Van Der Linden, S. Milyaeva, and A. Rashid, "Industry responses to the european directive on security of network and information systems (NIS): Understanding policy implementation practices across critical infrastructures," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, 2020, pp. 301–317.

[4] C. Kohler, "The eu cybersecurity act and european standards: an introduction to the role of european standardization," *International Cybersecurity Law Review*, vol. 1, no. 1, pp. 7–12, 2020.

[5] "ISO/IEC 15408-1:2009. Information technology — Security techniques — Evaluation criteria for IT security," https://www.iso.org/standard/50341.html.

[6] O. Potii, O. Illiashenko, and D. Komin, "Advanced security assurance case based on iso/iec 15408," in *International Conference on Dependability and Complex Systems*. Springer, 2015, pp. 391–401.

[7] "ISO/IEC 18045:2022. Information security, cybersecurity and privacy protection — Evaluation criteria for IT security — Methodology for IT security evaluation," https://www.iso.org/standard/72889.html.

[8] S. Dotsenko, O. Illiashenko, S. Kamenskyi, and V. Kharchenko, "Integrated model of knowledge management for security of information technologies: standards iso/iec 15408 and iso/iec 18045," *Information & Security*, vol. 43, no. 1, pp. 305–317, 2019.

[9] C. National Research Council, "Standards, conformity assessment, and trade into the 21st century," *StandardView*, vol. 5, no. 3, pp. 99–102, 1997.

[10] "Standards, conformity assessment and developing countries," *Policy Working Papers, World Bank, Washington, DC, May*, 1997.

[11] 2018 reform of eu data protection rules. European Commission. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[12] J. P. Albrecht, "How the gdpr will change the world," *Eur. Data Prot. L. Rev.*, vol. 2, p. 287, 2016.

[13] A. Michota and N. Polemi, "A supply chain service cybersecurity certification scheme based on the cybersecurity act," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2022, pp. 382–387.

[14] P. K. Infrastructure and T. P. Profile, "Common criteria for information technology security evaluation," *National Security Agency*, 2002.

[15] A. Fekete, "Common criteria for the assessment of critical infrastructures," *International Journal of Disaster Risk Science*, vol. 2, no. 1, pp. 15–24, 2011.

[16] G. Stoneburner, "Developer-focused assurance requirements [evaluation assurance level and common criteria for it system evaluation]," *Computer*, vol. 38, no. 7, pp. 91–93, 2005.

[17] A. Hunstad, J. Hallberg, and R. Andersson, "Measuring it security - a method based on common criteria's security functional requirements," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, 2004, pp. 226–233.

[18] G. Spanoudakis and T. Tsigkritis, "Towards a framework for dynamic verification of peer-to-peer systems."

[19] G. Spanoudakis, C. Kloukinas, and K. Mahbub, "The serenity runtime monitoring framework," in *Security and Dependability for Ambient Intelligence*. Springer, 2009, pp. 213–237.

[20] R. Kowalski and M. Sergot, "A logic-based calculus of events," in *Foundations of knowledge base management*. Springer, 1989, pp. 23–55.

[21] R. Kowalski and F. Sadri, "Reconciling the event calculus with the situation calculus," *The Journal of Logic Programming*, vol. 31, no. 1-3, pp. 39–58, 1997.

[22] M. Shanahan, "The event calculus explained," in *Artificial intelligence today*. Springer, 1999, pp. 409–430.

[23] E. T. Mueller, *Commonsense reasoning: an event calculus based approach*. Morgan Kaufmann, 2014.

[24] ——, "Event calculus reasoning through satisfiability," *Journal of Logic and Computation*, vol. 14, no. 5, pp. 703–730, 2004.

[25] W. Liu and M. Li, "Requirements planning with event calculus for runtime self-adaptive system," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 77–82.

[26] L. Liberti and F. Marinelli, "Mathematical programming: Turing completeness and applications to software analysis," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 82–104, 2014.

[27] "Drools RETEOO (Open Source Knowledge)," https://salaboy.com/2011/06/06/drools-reteoo-for-dummies-1-intro/.

[28] "Fair Isaac COrporation (FICO): RETE III)," https://www.fico.com/blogs/what-rete-iii.

[29] R. B. Doorenbos, "Production matching for large learning systems." Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science, Tech. Rep., 1995.

[30] B. Berstel, "Extending the rete algorithm for event management," in *Proceedings Ninth International Symposium on Temporal Representation and Reasoning*. IEEE, 2002, pp. 49–51.

[31] D. Liu, T. Gu, and J.-P. Xue, "Rule engine based on improvement rete algorithm," in *The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding*. IEEE, 2010, pp. 346–349.

[32] T. Patkos, D. Plexousakis, A. Chibani, and Y. Amirat, "An event calculus production rule system for reasoning in dynamic and uncertain domains," *Theory and Practice of Logic Programming*, vol. 16, no. 3, pp. 325–352, 2016.

[33] B. Venners, "The java virtual machine," *Java and the Java virtual machine: definition, verification, validation*, 1998.

[34] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1946–1956.

[35] "Elasticsearch: Filebeat overview," https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html#filebeat-overview.

[36] E. Rescorla, "The transport layer security (tls) protocol version 1.3," ., Tech. Rep., 2018.

[37] "Postgres Wiki Main Page Documentation," https://wiki.postgresql.org/wiki/Main_Page.

[38] B. Mustafa, W. Tao, J. Lee, and S. Thorpe, "Kubernetes from the ground up deploy and scale performant and reliable containerized applications with kubernetes," *Kubernetes from the ground up deploy and scale performant and reliable containerized applications with Kubernetes*, 2018.

[39] "DevOps for NetOps from Rick Sherman Puppet Labs," https://archive.nanog.org/sites/default/files/05-Sherman-StLouis.pdf.

[40] "AWS vs Azure vs Google Cloud: What is the best cloud platform for enterprise," https://www.computerworld.com/article/3429365/aws-vs-azure-vs-google-whats-the-best-cloud-platform-for-enterprise.html.

[41] J. Shah and D. Dubaria, "Building modern clouds: using docker, kubernetes & google cloud platform," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0184–0189.

[42] "Kubernetes Nodes cluster Architecture." https://kubernetes.io/docs/concepts/architecture/nodes/.

[43] "Kubernetes Jobs official documentation." https://kubernetes.io/docs/concepts/workloads/controllers/job/.

[44] "Cyber security 4.0: protecting the Industrial Internet Of Things(C4IIOT)," https://cordis.europa.eu/project/id/833828.

[45] "A coordinated framework for cyber resilient supply chain systems over complex ICT infrastructures(FISHY)," https://cordis.europa.eu/project/id/952644.

[46] "A Dynamic and Self-Organized Artificial Swarm Intelligence Solution for Security and Privacy Threats in Healthcare ICT Infrastructures(AI4Healthsec)," https://cordis.europa.eu/project/id/883273.

[47] "Intelligent, distributed, human-centered and trustworthy IoT environments(IntellIOT)," https://cordis.europa.eu/project/id/957218.

[48] "Certifying the Security and Resilience of Supply Chain Service (CYRENE)," https://cordis.europa.eu/project/id/952690.

[49] "RESilient transport InfraSTructure(RESIST)," https://cordis.europa.eu/project/id/769066.

[50] "Smart Big Data Platform to Offer Evidence-based Personalised Support for Healthy and Independent Living at Home(SmartBear)," https://cordis.europa.eu/project/id/857172.

[51] "A Comprehensive cyber-intelligence framework for resilient collabo-rative manufacturing Systems(COLLABS)," https://cordis.europa.eu/project/id/871518.

[52] E. Lakka, G. Hatzivasilis, S. Karagiannis, A. Alexopoulos, M. Athanatos, S. Ioannidis, M. Chatzimpyrros, G. Kalogiannis, and G. Spanoudakis, "Incident handling for healthcare organizations and supply-chains," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–7.