Master thesis : UniForeXT: A Unified Data Model for Cross-Tool Integration in Digital Forensic Triage

Auteur : Robert, Louan
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master en sciences informatiques, à finalité spécialisée en "computer systems security"
Année académique : 2025-2026
URI/URL : https://github.com/louanrobert/UniForeXT.git; https://zenodo.org/doi/10.5281/zenodo.20396207; http://hdl.handle.net/2268.2/26068


UNIVERSITY OF LIÈGE
School of Engineering and Computer Science

and

UniForeXT: A Unified Data Model for Cross-Tool Integration in Digital Forensic Triage

---

ROBERT LOUAN

Thesis presented in fulfillment of the requirements for the
Master's degree in Computer Science

Thesis supervisor:
PROF. BENOÎT DONNET

Academic year: 2025–2026

Acknowledgments

I would like to sincerely thank my supervisor, Prof. Benoît Donnet, for their guidance, support, and constructive feedback in this master thesis process. Their knowledge and encouragement were very useful in making this work possible.

I also want to thank Prof. Christophe Debruyne for their advice, availability, resources, and useful discussions that helped me in the implementation of this work. Although I did not follow the "Advanced Databases" and "Knowledge Representation and Reasoning" courses, I acquired the necessary background through independent study during this thesis, and these discussions were instrumental in that process.

Furthermore, I would like to thank the Center for Cybersecurity Belgium (CCB), where I had a great work atmosphere and the resources to conduct this work. Moreover, I would like to thank Grégoire Lambin for their supervision of this master thesis at the CCB, Joris Herbots for their encouragement and constructive discussions that greatly helped me complete this work and Hervé Jadot for their ideas and opinion. Overall, I also want to thank the people at the CCB for giving me this opportunity and for their support.

Additionally, I would like to express my deepest gratitude to my family, partner, and friends for their constant support, patience, and encouragement throughout my studies. Their confidence in me has been invaluable during this journey. This work would not have been possible without their encouragement and motivation.

Finally, I would like to thank all the teachers I had at the University of Liège, as well as throughout my primary and secondary school years, for their commitment to share their knowledge. These teachers played a significant role in shaping my education and personal growth.

AI tools disclosure

Parts of this thesis were improved using DeepL Write [1] to improve grammar and clarity. Some sentences were rewritten by LLMs to make them clearer or allow them to introduce notions more fluently.

The suggested changes were reviewed and were not considered to be the source of truth. Special care was taken to ensure that there were no changes to the meaning or content. The

tools were used solely for language refinement: all ideas, analyzes, and conclusions are the author's own.

LLMs were used as helpers in formatting this document for tables, listing styles, and improving the overall layout, etc.

Summary

Digital Forensics and Incident Response (DFIR) triage increasingly relies on heterogeneous outputs produced by multiple forensic tools. In operational contexts, such as at CERT Belgium, this heterogeneity creates a significant bottleneck. Tools produce heterogeneous results with different schemas and file formats that are difficult to correlate, query, and interpret under time constraints. Analysts must manually reconcile schema differences, duplicate entities, and inconsistent terminology before forming a coherent view of the incident. This thesis addresses this issue by proposing a unified, tool-agnostic approach to integrate forensic outputs into a single, large-scale, semantic representation.

This thesis is structured around two research questions. The first question explores which data model and modeling principles can unify outputs from heterogeneous forensic tools while preserving integration and queryability for triage. The second question is supplementary and investigates how a dedicated visualization interface can operationalize that model for analyst-driven investigations. To answer these questions, the thesis introduces **UniForeXT**, an ontology, a configuration-driven mapping pipeline that transforms tool outputs into knowledge graph instances, and an interactive visualization layer that supports temporal, structural, and query-based exploration.

To justify the modeling choice, relational, document, and graph logical models were compared against weighted criteria derived from functional and non-functional requirements. These requirements included semantic integration, cross-source integration, query expressiveness, provenance, and interoperability. The graph model performed consistently better than the alternatives, and a sensitivity analysis confirmed the robustness of this decision under weight perturbations and break-even analysis. Based on these results, UniForeXT was implemented as an ontology with RDFS semantics. It is organized around core forensic entities such as detections, events, processes, users, computers, and log files, as well as controlled value concepts such as attack tactics, techniques, and logon types. It includes explicit relationships, inverse links, and forensic tool provenance.

The mapping layer is implemented in Java as a generic semantic transformation pipeline. Rather than using hardcoded per-file logic, mappings are externalized in YAML configurations, which define identifier construction, class instantiation, property population, and reusable mapping fragments. This design improves extensibility and maintainability while enabling the addition of new tool outputs with no code changes. The design of this component is based on the Single Responsibility and Open-Closed principles. Together, these principles

allow for a codebase that is easy to maintain and extend. For example, adding support for new file formats requires no code changes (i.e, it simply requires an extension of the existing code). After the mapping stage, the resulting graph is validated with SHACL constraints before being serialized to RDF/Turtle. This ensures structural consistency while preserving flexibility for imperfect forensic datasets.

To operationalize the model for triage (RQ2), a desktop visualization application was developed using JavaFX and embedded HTML/JavaScript components. It provides complementary views: an event explorer with multiple charts on different aspects of the data, a high-density timeline for temporal analysis, a graph neighborhood explorer for contextual exploration, a SPARQL query interface for custom analysis, and an undated-items view. Together, these views support an analyst workflow that moves from overview to drill-down while remaining anchored in one unified model.

Validation was conducted using datasets provided by CERT Belgium, each containing outputs from multiple forensic tools. Competency questions were translated into SPARQL queries to test model answerability and requirement coverage. Core triage and provenance queries were successfully answered, demonstrating support for cross-tool integration, traceability, and interactive analysis. Performance results indicate operational feasibility: the mapping process completed in about 31.47 seconds per dataset and representative SPARQL queries executed in under 300 ms. Some MITRE ATT&CK semantics and processes queries could not be fully evaluated due to dataset limitations (missing extractable ATT&CK/process details), rather than ontological limitations.

Overall, this thesis demonstrates that combining a modeled ontology with configuration-based semantic mapping and purposely designed visualization can significantly reduce fragmentation in DFIR triage workflows. UniForeXT provides a practical and extensible foundation for integrating heterogeneous forensic outputs into a coherent, queryable, and analyst-oriented knowledge graph. Future work includes web-based multi-user deployment, stronger entity reconciliation for noisy identifiers, interoperability with standard KGC frameworks (e.g., RML/YARRRML), larger-scale benchmarking, and user studies with incident responders.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

In today's digital landscape, cyber events are becoming both more frequent and more sophisticated [2], generating a tremendous volume of forensic data that must be examined to understand what actually occurred. Several factors contribute to this increase, including the growing digitalization of services and companies, the adoption of cloud infrastructures, and the expanding attack surface of interconnected systems. The 2025 Hiscox Cyber Readiness Report, drew on insights from nearly 6,000 small and medium-sized enterprises (SMEs) across seven countries, found that 59% of respondents experienced a cyber attack in the last twelve months [3]. Modern cyber threats span a wide range of attack vectors. Among surveyed SMEs, 27% reported experiencing a ransomware attack, and 80% of those affected paid the ransom. Other major threats include supply chain attacks, which compromise trusted software dependencies to infiltrate systems, and advanced persistent threats (APTs), which can remain undetected within networks for long periods while continuously extracting data or disrupting operations. Most system vulnerabilities and their impacts are not predictable, making it tough for those who run a company to define possible attacks [4]. These incidents can have significant financial, operational, and societal impacts, including service disruptions, data breaches, reputational damage, and regulatory consequences. The Hiscox report found that a third of attacked SMEs received a substantial fine following an incident [3]. In the 2020 Global Risks Report [5] by the World Economic Forum, cyberattacks were listed as the seventh most likely and the eighth largest impact risk among all risk categories.

As these threats evolve, the amount of data (e.g., logs, metrics, etc) needed to trace and detect malicious activity by information systems has increased substantially [6]. Logging, tracing, and artifact creation for endpoints, servers, network devices, and cloud platforms happen at a constant rate, thoroughly capturing system behavior. Although such a valuable amount of data is critical in any forensic analysis process, the sheer volume and variety of data make it increasingly challenging to extract useful insights from them. In reality, gaining insights from the collected data is not merely about basic data examination. This includes log inspection and correlation, timeline tracing, and low-level artifact examination.

To address these issues, different automated analysis tools have been developed to help analysts extract characteristics and identify any suspicious activities.

Hayabusa [7] is one of them. Developed by the Yamato Security group in Japan and written in Rust, it is an open-source Windows event log fast forensics threat hunting tool. It leverages

Sigma-based detection rules to scan `.evtx` log files and consolidate results into a structured timeline, enabling analysts to efficiently triage large volumes of Windows event logs and identify suspicious activity.

Sigma [8] is an open-source, generic standard for creating and sharing threat detection logic. Each rule captures the log source and detection logic using YAML. As a result, a single Sigma rule can be used across multiple tools and security systems. This makes Sigma for log files what Snort is for network traffic and YARA is for files.

Another tool is Chainsaw [9]. Developed by WithSecure, it is an open-source tool designed for rapid triage of Windows forensic artifacts, particularly event logs. It offers a generic and fast method of searching through event logs for keywords and identifying threats using custom Chainsaw and Sigma detection rules. Results can be exported in various formats, including ASCII table, CSV, and JSON.

However, these tools often have specific detection methodologies and produce specific data formats. This results in fragmented outputs that lack a unified interpretation. Although there are numerous methods capable of analyzing data automatically and identifying inconsistencies, most of them produce extensive results that need further verification by a human analyst. Many of these outputs turn out to be false alarms, duplicate warnings, and irrelevant clues with no practical significance. Hence, much effort has to be invested by analysts in filtering, verifying, and enriching this information before making sense of it. This creates more work for the analyst rather than providing him with the necessary assistance.

These challenges are encountered within the broader context of Digital Forensics and Incident Response (DFIR), which is a discipline within cybersecurity that focuses on the identification, investigation, and remediation of security incidents. It combines forensic analysis techniques with operational response strategies to both understand the root cause of an incident and limit its impact on affected systems. DFIR plays a critical role in modern organizations, where the ability to quickly detect and respond to threats directly influences the overall security posture [10, 11, 12].

The DFIR methodology is usually divided into several phases [10]. The identification phase consists of detecting potential security incidents through monitoring systems and forensic analysis. Once an incident is confirmed, the containment phase aims to limit its spread, for instance, by isolating compromised systems or disabling affected accounts. This is followed by the eradication phase, during which the root cause of the incident is removed, such as deleting malware or closing exploited vulnerabilities. The recovery phase ensures systems return to full operational capacity while preventing reinfection. Finally, the lessons learned phase involves documenting the incident and improving defensive mechanisms to better handle future threats [12].

Within this process, triage constitutes a crucial preliminary step that directly impacts the efficiency of the overall response. During triage, analysts must rapidly assess a potentially large number of systems to determine which ones have been compromised and require immediate attention. This phase is particularly challenging in large-scale environments, where incidents may affect dozens or even hundreds of endpoints simultaneously. The objective is not to perform a complete forensic investigation on each system, but rather to prioritize resources by identifying the most critical cases [12].

The complexity of triage is further increased by the intense pressure regarding time that is inherent to digital forensics and incident response. Failure to detect and contain compromised hosts may lead to further compromise through the spread of malware, data destruction or encryption, data breaches, and destruction of evidence. The issue is made even more complex by the vastness and diversity of information acquired through investigations. The need to quickly interpret diverse forensic artifacts while maintaining a high level of confidence in the results highlights the importance of efficient data processing and analysis techniques [12].

As part of their duties, DFIR investigators are charged with handling remediation incidents in varied infrastructures ranging from physical computers to cloud based infrastructure servers. Therefore, an incident usually affects many different types of systems depending on the size of the network and severity of the incident.

Due to the diversity of digital evidence collected from many different varieties of systems, incident responders quickly analyze endpoints to identify potentially affected devices. This is the first step in prioritizing remediation actions. They do this by collecting various important logs, file metadata, processes, and network traffic using collection software such as the Kroll Artifact Parser and Extractor (KAPE). The collected digital evidence can be analyzed using various kinds of commercial and non-commercial software. Despite the aid of automated detection methods, such as Sigma rules, efficiently handling these artifacts is often a manual task for DFIR teams.

In practice, when CERT Belgium[1] is conducting an incident response and investigation, the relevant system administrators are tasked with running a KAPE extraction on all devices present in the network and then uploading the extraction results to a secured object storage. After obtaining the extraction results, a CERT customized pipeline uses different forensic tools to analyze the log files, where each tool provides results in its own format, resulting in heterogeneous outputs that do not give a comprehensive view of the device. Some tools might provide detection alerts, while others provide all the actions taken on the endpoint (file system, user sessions, executions, configuration changes, etc).

[1] Cyber Emergency Response Team, CERT Belgium is part of the CCB

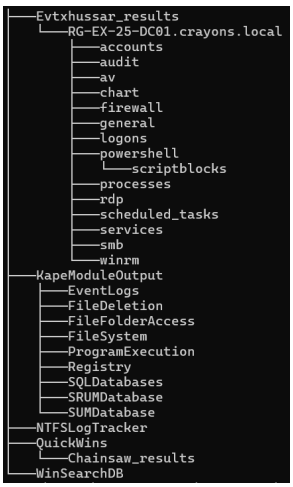


Figure 1.1.: Overview of the root results of the forensics tools, only directories are displayed

As illustrated in Figure 1.1, the use of multiple forensic tools, each tailored to specific detection methodologies, leads to a highly fragmented output. While these tools are effective individually, their combined use in large-scale investigations raises important challenges regarding data integration and analysis.

Multiple tools generate results in different formats, including structured tabular data (CSV, XLSX), unstructured scripts (PowerShell), and tool-specific artifacts. Besides, each tool has its own structure for organizing the output, resulting in inconsistencies in naming conventions, data schema, and abstractions.

For example, authentication-related events may be distributed across various outputs. There may be separate files for logins, remote SSH, RDP, among others. Meanwhile, process execution traces may appear in both dedicated process analysis outputs and lower-level file system artifacts. This fragmentation makes it difficult for analysts to correlate events across tools and create a coherent timeline of the incident.

As a result, DFIR analysts are confronted with heterogeneous data complicating the construction of a coherent incident overview and often requiring manual intervention or tool-specific expertise for triage decisions. Performing triage decisions requires manual intervention or tool-specific expertise to build a coherent overview of an incident. This situation increases analysis time, introduces errors, and limits scalability when working with a large number of systems.

This master thesis is motivated by the operational needs of CERT Belgium and aims to address the challenges of multiple forensic tool outputs by proposing a framework that facilitates the integration and exploitation of heterogeneous forensic data for large-scale incident triage. This requires the creation of a unified data model to consolidate heterogeneous data into a

single common model and the selection of an on-disk storage format. This work also creates a mapping tool used to map heterogeneous outputs to the unified data model. Finally, a visualization interface allows analysts to explore the instantiated model.

To address these challenges, two research questions arise:

> **RQ1:** "*What data model and associated modeling principles can effectively unify outputs from heterogeneous forensic tools to enable efficient cross-tool integration and querying for large-scale incident triage?*"
>
> **RQ2 (subsidiary):** "*How can a purpose-built visualization interface operationalize this model to support efficient, analyst-driven triage at a large scale?*"

To answer these research questions, this thesis makes three contributions. *UniForeXT*, a unified data model for heterogeneous forensic tool outputs. A mapping tool that will map the heterogeneous outputs to the unified data model. A visualization tool that will load the instantiated data model to provide means of visualization for the analysts.

The remainder of this master thesis is organized as follows. Chapter 2 presents the background and foundational concepts that underpin this work. Chapter 4 reviews related work and existing approaches. Chapter 3 formalizes the problem statement and research questions. Chapter 5 introduces the UniForeXT model, its logical model selection and its modeling decisions and rationale. Chapter 6 describes the mapping tool and data ingestion pipeline used to transform heterogeneous outputs into an instantiated model. Chapter 7 presents the visualization interface for analyst-driven triage and its implementation details. Chapter 8 evaluates the proposed approach. Chapter 9 concludes the thesis and outlines directions for future work.

Chapter 2 Background

This chapter establishes the technical and methodological foundations upon which the rest of this thesis is built. Effective incident response depends on the ability to collect, interpret, and reason over evidence left behind by an attacker. That process rests on three interrelated pillars. Logging determines how system activity is recorded and its content. Digital forensics provides the scientific framework for acquiring and analyzing evidence. Log analysis turns raw records into actionable conclusions.

2.1. Logging

Logging refers to the process of recording events and activities within a software or hardware system to capture its runtime behavior. In practice, this process extends beyond simple application logs. It includes a wide range of telemetry. This includes system logs, application logs, network traces (e.g., packet captures or PCAPs), performance metrics (CPU, memory, disk I/O), system health indicators, security and audit logs, and even distributed traces in complex architectures. Typically, this process begins with developers inserting log statements directly into the source code but also involves configuring external logging and monitoring tools across the infrastructure. These records generally include a log verbosity level (`info`, `debug`, or `error`), a timestamp, contextual metadata (e.g., host, service, and request ID), and event-specific information. This information enables comprehensive observability, troubleshooting, and analysis of system behavior [13].

Logs can originate from multiple layers of a system, leading to a high degree of heterogeneity. Common categories include application logs, operating system logs, security logs, and network logs. Each category follows a different schema and format. For instance, application logs are often unstructured or semi-structured text, while modern systems increasingly adopt structured formats such as JSON. In contrast, Windows Event Logs use a binary format (`.evtx`) with a predefined schema. This diversity complicates automated processing, as each format requires specific parsing and normalization strategies.

The existence of logging is primarily motivated by the need for observability and accountability within software systems. During development, logs help programmers debug their applications by offering insights into how the code executes, which paths are taken, and where failures or unexpected behaviors occur. In production environments, logging becomes

even more critical: it allows for monitoring system health, detecting anomalies, tracking usage patterns, and diagnosing issues without directly interfering with the live system. Furthermore, logging supports auditing and compliance requirements, as organizations can maintain detailed records of system operations and user interactions, which may be legally or operationally necessary [14].

Typically, logs are written to persistent storage like files, databases, or centralized logging systems, making them accessible for monitoring and later analysis. Logs are present at multiple locations on complex systems. On computers, all the possible interactions would lead to great variety of logs in terms of originator, schema, precision, format, location, etc. [13].

From a broader perspective, logging contributes to software reliability, maintainability, and security. By providing a transparent view of system activities, logs help prevent errors from going unnoticed, enable quick responses to incidents, and offer a historical context that is essential for root cause analysis. Effective logging strategies often balance verbosity and clarity, ensuring that logs provide meaningful information without overwhelming storage or processing resources. Overall, logging is a fundamental practice in software engineering, bridging the gap between abstract code and real-world system behavior.

Log management is a common challenge in incident response. Organizations often underestimate its importance and fail to allocate enough resources to comprehensively collect, centralize, and retain logs across their infrastructure. Consequently, critical data from network devices, servers, endpoints, and applications, such as system events, security logs, and network traffic traces, may be incomplete, inconsistent, or entirely absent. This lack of visibility significantly hinders the ability to detect, investigate, and respond to security incidents effectively [12].

2.2. Digital Forensics

Since the generalization of computer devices, digital forensics has become increasingly important in cases where digital devices are used to commit crimes [15]. Digital forensics is defined as the application of scientific methods for the identification, preservation, collection, and analysis of digital evidence to reconstruct past events and support legal proceedings. Modern investigations generally classify the role of digital devices into three distinct categories: as a repository of evidence, as a tool to commit a crime, or as the target of an attack [10, 16].

The device as a repository or a tool: criminal-owned When a device is owned or operated by a suspect, digital forensics aims to reconstruct the history of digital media to establish intent and premeditation [17].

- Behavioral profiling: investigators analyze digital traces such as search history, communication logs (e.g., emails, chat logs, etc.), and location data to understand the suspect's behavior
- Timeline reconstruction: forensic analysis is used to establish a chronological sequence of events, helping to verify alibis, events, truths, and possibly locations
- Evidence recovery: it involves the retrieval of hidden, ciphered, or deleted files that may serve as direct evidence of a crime, such as financial records in fraud cases, illicit data, or any other evidence

The device as a target: incident response When a device and/or network is attacked, digital forensics is integrated with Incident Response to mitigate damage and identify the modus operandi of the attacker [10, 16]:

- Vulnerability assessment: investigators determine how the attack occurred by identifying the security breach used during the "Weaponization" and "Delivery" phases of the attack
- Methodology: forensics identifies the tools and techniques used for exploitation, installation of persistence, and command-and-control communication
- Attribution and responsibility: the goal is to answer the "5Ws and How" (Who, What, When, Where, Why, and How) to determine the scope of the impact and provide evidence suitable for insurance claims, regulatory compliance, or criminal prosecution.

This master thesis will focus on this case, where the device is considered a target.

2.3. Collection Strategies

The approach to data collection has evolved from a "collect as much as possible" model to a risk-based strategy. The choice between a light and a complete acquisition depends on the required speed of the investigation, the amount of possible infected endpoints, and the availability of the evidence (cloud computing, etc). Live data acquisition is used for volatile data and digital evidence present inside routers, chat rooms, RAM, etc. Live data acquisition may require specific tools and should be identified prior to operations [18].

When dealing with incidents involving many endpoints, analysts must handle a large volume of data. This significantly impacts both the scale, the size, and the time required to process the information from the investigation. In incident response, time is critical, as delays can cause further damage or additional downtime.

Triage is a targeted collection strategy used to identify high-value evidence quickly. Instead of imaging an entire drive, investigators focus on specific artifacts of interest, primarily system logs, memory-resident artifacts, and configuration files. It quickly determines whether an endpoint is compromised [19]. Triage is also used to prioritize during incident response and mitigation.

Nevertheless, complete collections are still important. They can be done in multiple ways depending on the system state. If the system is live, the RAM and other volatile elements can be extracted for analysis. On dead systems, only non-volatile elements can be captured. For the capture of elements, there are two techniques:

- The bit-stream copy copies every bit present on the storage medium. This includes all allocated space, unallocated space (deleted files), and slack space [18].
- The logical image only captures the referenced objects by the file system or OS. It is faster but misses hidden data like slack space or deleted fragments [17, 18].

These three collection strategies provide advantages and disadvantages. While the bit-stream copy allows for a complete collection, which gives maximum evidence, it is slow and expensive in terms of computing resources and can sometimes be infeasible. On the other hand, the logical image is still expensive but depends heavily on the usage of storage resources. For example, on a 1TB drive used at 10%, the logical image will only capture the 100GB of referenced objects, whereas the bit-stream copy will capture the 1TB of data. Due to its nature of capturing only the files of interest, it allows for a very quick capture and can even be predictable. The fastest acquisition technique is targeted collection. By restricting the scope to predefined artifacts of interest, it minimizes the amount of data collected, enabling rapid acquisition with more predictable performance. However, this approach inherently risks omitting relevant evidence outside the predefined scope.

Regardless of the strategy used, ensuring the integrity of the extracted data is a critical requirement in forensic investigations. To guarantee that the collected artifacts have not been altered (whether intentionally or not), analysts rely on cryptographic hash functions (such as SHA-256). A hash function returns a fixed-size digital fingerprint of the data it receives. The fingerprint is cryptographically tied to the original data: even a single-bit modification results in a completely different hash fingerprint, making it possible to detect any alteration of the original data. During extraction, hash values are computed before and after acquisition,

and they are regularly verified throughout the analysis process to ensure evidence integrity. This mechanism also supports the chain of custody, providing formal proof that the evidence used in a criminal investigation is identical to the original found data [18].

2.4. Log Analysis for Digital Forensics

To determine possible cyber attacks and endpoint compromises (a network could be partially compromised), an analyst uses log files to determine the actions taken on an endpoint. Analyzing log files by hand would be almost impossible due to the large volume of logs generated by complex systems. Log analysis methods are used to transform the recorded logs into meaningful information [14].

The most basic method to perform log analysis is to use regular expressions to search for and/or parse log entries. This approach requires the definition of regular expression (regex) rules for every log file. The number of rules and their complexity can be affected by system configuration and/or version due to variations in log file format [14, 20].

Modern digital forensics increasingly relies on specialized tools that automate log parsing, detection, and correlation. To perform log analysis, CERT Belgium is using, among others, Chainsaw and Hayabusa, which are widely used for the analysis of Windows event logs, and Sigma, which provides a standardized way to describe detection rules [11].

Chainsaw [9] is a fast log analysis and threat hunting tool designed to work primarily with Windows Event Logs (.evtx files). It enables investigators to quickly search through large volumes of logs using keyword matching, regular expressions, and pre-defined detection rules. One of its key features is the ability to apply Sigma rules directly, allowing analysts to detect known attack patterns without manually crafting complex queries.

Hayabusa [7] extends this approach by focusing on high-performance timeline generation and advanced detection capabilities. It parses Windows event logs and correlates events to produce a chronological view of system activity, which is essential for reconstructing attack sequences. Hayabusa also integrates Sigma rules but enhances them with additional filtering, tagging, and mapping to frameworks such as the MITRE ATT&CK model. This makes it especially effective for identifying suspicious behaviors such as lateral movement, privilege escalation, or the use of administrative tools.

Sigma plays a central role in modern log analysis by providing a generic and portable rule format for log-based detection. Instead of writing tool-specific queries or regular expressions, analysts can define detection logic in Sigma's structured YAML format. These rules can then be converted into queries compatible with various tools and platforms, including Chainsaw and

Hayabusa. This abstraction simplifies rule sharing and reuse across different environments and improves consistency in threat detection.

Chainsaw, Hayabusa, and other log analysis tools process the raw logs in a similar manner. Their first step is to parse the log files they are interested in and are able to understand. During the parsing, they read and extract fields of interest from multiple files that could have different formats (syslog is plain text, .evtx files are binary, etc.) [21]. The second step is to normalize these extracted fields to a unique schema. Log files could have different naming conventions for some fields (for an IP address, it could src_ip, ip_address, clientIP, etc). The third step is the integration of the entries to link multiple log events together to identify patterns or behaviors.

Chapter 3 Problem Definition and Requirements

Incident response begins with a triage phase in which analysts must quickly estimate the scope, impact, and severity of a potential security incident. This phase depends on a limited set of high-value forensic artifacts (e.g., logs, process metadata, persistence indicators, and system configuration) collected under strong time and resource constraints.

In practice, these artifacts are processed by a pipeline of specialized forensic tools. Each tool targets specific artifact types and produces output in its own format and abstraction level. Consequently, analysts must work with heterogeneous outputs that differ in schema, terminology, and semantics.

This heterogeneity causes two operational issues:

- manual cross-tool correlation is slow and error-prone
- inconsistent naming and structure reduce the reproducibility and reuse of analysis results

This thesis addresses these limitations through two contributions. First, a unified data model designed to represent and integrate heterogeneous forensic outputs (RQ1). Second, a visualization interface that operationalizes this model to support triage (RQ2).

3.1. Functional Requirements

Data model (RQ1)

For the developed model (RQ1), six functional requirements are defined.

- **FR1 - Multi-source ingestion:** requires that the model shall ingest artifacts produced by multiple forensic tools and data sources.
- **FR2 - Unified representation:** requires that the model shall map heterogeneous tool outputs to a common schema with explicit semantics.
- **FR3 - Integration support:** requires that the model shall support integration of related entities and events across tools, including partially matching data.

- **FR4 - Queryability:** requires that the model shall support efficient querying for common triage questions (who, what, when, where, how).
- **FR5 - Visualization readiness:** requires that the model shall expose relationships in a form suitable for visualization at the model level.
- **FR6 - Traceability:** requires that the model shall preserve provenance links from normalized elements to original tool outputs.

Visualization interface (RQ2)

For the visualization interface (RQ2), three functional requirements are defined.

- **FR7 - Cross-tool navigation:** requires that the interface shall allow analysts to navigate and explore entities and events across tool boundaries from a single view.
- **FR8 - Triage-oriented filtering:** requires that the interface shall support filtering and scoping (e.g., by time range, endpoint, etc.).
- **FR9 - Visualization capabilities:** requires that the interface allow analysts to visualize the model through a timeline, charts (e.g., histograms, boxplots, etc.).

3.2. Non-functional Requirements

Eight non-functional requirements govern the architecture and quality of the model.

- **NFR1 - Tool-agnostic design:** requires that the architecture shall support the integration of new tools without modifying the core conceptual model.
- **NFR2 - Extensibility:** requires that the schema shall allow the addition of new artifact and entity types with minimal impact on existing mappings.
- **NFR3 - Interoperability:** requires that the model shall rely on explicit, documented semantics to reduce ambiguity across tools.
- **NFR4 - Performance for triage:** requires that data transformation and querying shall be fast enough to support operational time constraints (e.g., less than 5 seconds for a typical heavy query).
- **NFR5 - Data quality robustness:** requires that the model shall tolerate incomplete, noisy, and partially inconsistent forensic data.

- **NFR6 - Model quality:** requires that the model shall follow established modeling best practices, including appropriate denormalization and avoidance of common schema pitfalls.
- **NFR7 - Model correctness:** requires that the model shall be representative of the forensic data it must ingest and be competent in answering typical triage queries.
- **NFR8 - Data serialization:** requires that the instantiated unified model shall be serializable to a static, self-contained format suitable for cold storage on object storage, with no dependency on a live DBMS at query or read time.

3.3. Assumptions and Scope

Assumptions

The following assumptions support the design and scope of this work:

- Input data is legally and operationally accessible.
- Source tools provide machine-readable outputs and minimal metadata (timestamps, host identifiers, event context).
- Triage analysts can validate high-priority findings produced from correlated data.
- Heterogeneous artifacts are stored on object storage.

In scope

This project addresses the following aspects:

- Definition of a unified, tool-agnostic forensic data model.
- Normalization and integration of heterogeneous outputs for triage.
- Query and visualization support at the model level.
- Demonstration/evaluation on representative forensic datasets.
- Design and evaluation of a triage-oriented visualization interface built on the unified model (RQ2).

Out of scope

The following areas are explicitly excluded from this project:

- Full automation of incident response decision-making.
- Development or replacement of existing forensic collection tools.
- Deep forensic investigation beyond initial triage.
- Legal attribution and prosecution workflows.

3.4. Summary

Modern incident response triage requires analysts to rapidly assess the scope and severity of security incidents using forensic artifacts collected under tight time constraints. The core challenge is that these artifacts are processed by multiple specialized tools, each producing output in its own format, schema, and terminology. This heterogeneity creates two concrete problems: manual cross-tool correlation is slow and error-prone, and inconsistent naming reduces reproducibility.

The first contribution of this thesis is a unified, tool-agnostic forensic data model that normalizes and integrates these heterogeneous outputs (RQ1). The model is governed by six functional requirements: ingesting multi-source artifacts (FR1), mapping them to a common schema (FR2), supporting cross-tool integration including partial matches (FR3), enabling efficient triage queries (FR4), exposing relationships for visualization (FR5), and preserving provenance back to original tool outputs (FR6). Eight non-functional requirements further shape the architecture: tool-agnostic design (NFR1), extensibility (NFR2), semantic interoperability (NFR3), performance suited to operational triage timelines (NFR4), robustness to incomplete or noisy data (NFR5), adherence to modeling best practices (NFR6), correctness against real forensic data and typical triage queries (NFR7), and instantiated model serialization (NFR8).

The second contribution is a visualization interface built directly on the unified model (RQ2), governed by three functional requirements: cross-tool navigation (FR7), triage-oriented filtering (FR8), and timeline and chart visualization capabilities (FR9). The interface serves as applied validation of the model, demonstrating that its structure and semantics are sufficient to support practical triage workflows.

Chapter 4 Related Work

Existing literature in digital forensics explores various methods to handle the increasing volume of digital evidence. Researchers have proposed multiple frameworks to normalize and analyze this data. Many existing approaches rely on ontologies to model digital artifacts and to structure knowledge graphs representing investigative information. Some ontologies also encompass the full investigation process, allowing investigators to retrace and validate all steps taken during the investigation [22]. These ontology driven models aim to build a shared conceptualization of cyber incidents, allowing automated processes to reason over the relationships between users, objects, and events [23, 24]. Using standard models (i.e., models that are either commonly established or shared only between recipients) allows multiple entities to share evidence in an interoperable format during an investigation or evidence exchange [24].

? Ontologies and knowledge graphs

A knowledge graph is a graph data model in which nodes represent real-world entities and edges represent relations between them. Beyond simply storing data, knowledge graphs are designed to accumulate and convey knowledge.

A core challenge in any knowledge graph is ensuring that the terms used carry consistent and well-defined meaning. Ontologies are a solution to this problem. An ontology is a formal, explicit representation of the concepts and relations within a domain. Said otherwise, they are the shared convention on what terms mean. Ontologies also enable other features, such as reasoning.

Together, ontologies and knowledge graphs form a complementary pair: the graph provides a flexible, large-scale data structure, while the ontology provides the semantic layer that makes that data interpretable, interoperable, and amenable to automated reasoning [25].

Although existing ontological approaches largely focus on modeling raw artifacts directly, they primarily emphasize structured representation and semantic alignment of forensic evidence rather than the post-hoc harmonization of outputs from multiple cross-forensic tools.

4.1. Ontology Representations of Digital Evidences

The increasing reliance on a diverse set of proprietary and open-source tools to process heterogeneous digital evidence has highlighted the need for standardized data representations in digital forensics. Early systems, such as the Forensics of Rich Events framework [26], use knowledge graphs to represent the state changes of objects over time, though they often lack the semantic coverage required for complex cases. Subsequent frameworks, like the Semantic Analysis of Digital Forensic Cases [27], integrate knowledge management and mapping operators to reconstruct incident timelines and automatically measure correlations between subjects, objects, and events [23].

Within the broader cybersecurity domain, standardized formats have been heavily developed for sharing Cyber Threat Intelligence. Frameworks such as OpenIOC, VERIS, and the Structured Threat Information Expression (STIX) [28] enable the automated exchange of threat indicators and adversary behaviors [12, 29]. However, STIX and its predecessor CybOX [28], were fundamentally designed for threat intelligence and attack pattern detection, which consequently limits their expressivity when representing the broad spectrum of digital forensic traces [30]. Furthermore, STIX primarily captures associative relationships (i.e., what tools an adversary uses) but lacks the procedural semantics, preconditions, and environmental context required for forensic reconstruction [31].

To bridge the gap between threat intelligence and forensic science, the community developed the Unified Cyber Ontology (UCO) [32] and its domain-specific extension, the Cyber-investigation Analysis Standard Expression (CASE) [30, 33]. CASE serves as an open specification language designed specifically to advance coordinated cyber-investigations. By aligning with UCO, CASE provides a common structure to represent digital traces, investigative actions, and chain-of-custody provenance[30, 31]. CASE adds support for duck-typing[1]. A primary motivation of CASE is to enable the automated normalization, correlation, and exchange of forensic information across different organizations and evidence sources [30, 33].

NORIA-O [34], a recent ontology for anomaly detection and incident management in ICT systems, is used by Orange[2] to model complex relationships between network infrastructure characteristics, network activity and operations, and management activities. The implementation of the ontology created by its authors in the systems of Orange is underway [35].

[1] Duck-typing defines objects by their behavior and not their type. Duck-typing focuses on method at runtime rather than formal types declaration.
[2] A French telecommunication provider.

More recently, new ontologies have been created for specialized environments and advanced analytical workflows. For instance, the Smart City Ontological Paradigm Expression represents technology agnostic digital forensic evidence and threats specific to smart city infrastructures [36]. Other approaches leverage Digital Forensic Knowledge Graphs to integrate Large Language Models with digital forensics in a legal context [37].

Despite the fact that these existing standards provide a robust foundational vocabulary for digital artifacts, achieving seamless cross-tool integration remains a challenge due to the lack of standardization across native tool outputs. Although ontologies like UCO/CASE are highly expressive, effectively correlating the diverse logs, metadata, and structured reports generated by multiple distinct forensic suites into a single cohesive model requires dedicated aggregation frameworks [30]. Therefore, this work builds upon the principles of standard forensic ontologies by proposing a unified, cross-forensic-tool model that directly aggregates and harmonizes multiple disparate tool outputs, enabling automated evidence correlation and a shared, unified view of digital artifacts during investigations.

4.2. SIEM Systems

Security Information and Event Management (SIEM) systems are centralized platforms designed to aggregate, retain, and correlate diverse log and event data from multiple network security controls. By standardizing heterogeneous high-volume data formats into a unified representation, SIEMs allow security teams to perform routine analyzes, run centralized queries, and receive automated alerts based on preconfigured detection rules [12].

SIEMs rely heavily on rigid, rule-based logic and pattern matching to filter real-time security events [38]. SIEM event correlation is usually limited to predefined rulesets or algorithms. These require continuous, labor-intensive manual configuration and tuning to remain effective [12].

Although SIEM platforms excel at operational threat detection and log aggregation, there are several critical reasons why they are not a viable solution for creating a unified model to harmonize results from multiple forensic tools:

- Lack of semantic expressivity for deep forensics: SIEMs are optimized to process operational data such as network traffic logs, authentication events, and intrusion alerts. They lack the granular semantic expressivity necessary to accurately model the complex, highly specialized traces extracted by digital forensics suites, such as file system metadata, memory artifacts, or mobile application structures [30, 12].

- Rigid dependency on vendor-specific schemas: SIEM systems must constantly parse incoming data using unique, vendor specific schemas. This makes their effectiveness highly dependent on the availability of specific parsers and rules [29, 38]. If a forensic tool produces unstructured, ambiguous, or novel outputs, a SIEM will struggle to automatically normalize and interpret this data without extensive manual configuration. Conversely, ontological models use flexible, technology-agnostic structures (such as "duck typing" or extensible property bundles) that dynamically accommodate the diverse nature of forensic traces [30].
- Rule redundancy and alert fatigue: as organizations integrate more data sources into a SIEM, the underlying rule sets frequently become bloated with overlapping logic and redundant conditions, which degrades computational performance and causes severe alert fatigue for analysts [38]. Relying on a SIEM's rule engine to harmonize disparate and highly detailed forensic tool outputs would exacerbate these inefficiencies. A unified Knowledge Graph, however, systematically deduces relationships and consolidates entities based on deterministic identifiers and semantic reasoning, bypassing the inherent limitations of static rule-based correlation [39].

Therefore, while SIEMs are indispensable for initial incident detection and enterprise log management, their design constraints make them less suitable for preserving the rich semantic relationships and provenance information required in cross-tool forensic workflows.

4.3. Summary

Table 4.1 summarizes the key properties of the approaches surveyed in this chapter in relation to the requirements of forensic triage.

Although the existing landscape offers strong theoretical foundations, particularly CASE, no single approach simultaneously addresses cross-tool normalization, efficient querying at triage scale, and tool-agnostic extensibility for open-source DFIR pipelines. The comparison above highlights the trade-offs between expressive ontologies, operational log management systems, and more narrowly scoped exchange formats.

Table 4.1.: Comparison of related approaches against triage requirements

Approach	**Heterogeneous digital traces support**	**Cross-tool normalization**	**Multi-tools[1] output aggregation**	**Tool agnostic**	**Chain of custody**
FORE [26]	✓	×	×	✓	×
SADFC [27]	✓	×	×	✓	×
STIX [28]	×	×	×	✓	×
CybOX [28]	×	×	×	✓	×
UCO [32]	✓	×	×	✓	✓
CASE [33]	✓	×	×	✓	✓
NORIA-O [34]	✓	×	×	×	✓
SIEM platforms	✓	×	×	×	Recov[2]

[1] The scope is forensics analysis tools

[2] Recov: Recoverable, meaning some process needs to happen to get the full chain and that it is not included in the model

Chapter 5 UniForeXT: A Unified Model for Cross-Tool Integration in Digital Forensic Triage

The model name **UniForeXT** combines two components: *UniFore*, derived from **Uni**fied **Fore**nsics, and *XT*, where **X** serves as common shorthand for *cross* and **T** stands for *tool*.

In order to unify heterogeneous outputs produced by forensic tools, this thesis defines a shared model in which artifacts, events, and entities are represented with explicit semantics and linked provenance. The objective is to make cross-tool outputs understandable, queryable, and reusable for triage. This chapter directly addresses RQ1 (see Chapter 1).

The discussion in Chapter 4 showed that existing approaches provide important foundations, but no single solution simultaneously offers cross-tool normalization, triage-oriented querying, tool-agnostic extensibility, and explicit provenance in a form that remains practical for operational use. For that reason, this chapter first discusses the selection of a logical model to represent the unified model. It then presents **UniForeXT**, an ontology used as the semantic backbone of a knowledge graph.

5.1. Logical Model

Prior to defining a data model, a logical model should first be selected. This decision is based on the requirements set earlier (see Chapter 3).

5.1.1. Decision criteria

Five main criteria guide the decision of which model to pick, each based on functional and non-functional requirements. Each criterion is assigned a weight between $[0, 1]$ that reflects its importance based on the requirements. As will be discussed in the following subsections, these weights will be used later to calculate a weighted score for each candidate model. This score will ultimately drive the final selection.

The weights reflect the relative importance of each criterion according to the earlier defined requirements. Cross-source integration (C2) is assigned the highest weight (0.28) because it enables direct reconstruction of incident scenarios and allows for easier cross-tool analysis.

Table 5.1.: Decision criteria for logical model selection

Criterion	Weight	Description	Requirement(s)
C1	0.22	Semantic integration (normalization)	FR1, FR2, NFR2
C2	0.28	Cross-source integration (link related entities that span different tools)	FR3
C3	0.17	Query expressiveness for triage questions (5Ws & How)	FR4
C4	0.17	Provenance and traceability	FR6
C5	0.16	Semantic interoperability	NFR1, NFR3

Semantic integration (C1) is critical as well (0.22) due to the heterogeneity of forensic tool outputs. Query expressiveness (C3) and provenance (C4) are equally important (each with a weight of 0.17) because they support analyst reasoning and traceability of the artifacts (from which tool they originate). Finally, semantic interoperability (C5) is important for extensibility, reuse and integration with a mapping layer, but it is less critical than the other criteria, which is reflected in its slightly lower weight (0.16).

These weights reflect the reasoning detailed above and have been reviewed and approved by the CERT. This approach provides a practical and straightforward way to show why this model fits the problem. Furthermore, as already discussed in Chapter 4, other studies that have faced similar data challenges have turned to ontologies. This provides insight into, and perhaps confirmation of, the correct model.

This weighted scoring method is based on multi-criteria decision-making [40].

5.1.2. Comparison of candidate logical models

Three primary logical models are considered for representing forensic data: relational, document, and graph.

In relational models, data is stored in normalized tables with explicit references made to establish foreign keys. Such models have a rigid schema that is defined during the design. Join operations are explicitly performed to move across relationships. On the other hand, document models store semi-structured, nested data with an ever-changing schema. This schema is stored in the documents themselves, where attributes and relationships are implicit. Graph models represent entities as nodes and relationships between nodes as edges inside a graph. This makes graphs treat relationships as first-class citizens alongside entities [41].

5.1.3. Logical model evaluation

The criteria are evaluated qualitatively, based on the inherent properties of each logical model and independent of physical storage concerns. Scores are assigned on a scale of 0 to 3, where 0 indicates a poor fit and 3 indicates a strong fit.

Semantic integration (C1) [42, 43] identifies fundamental differences in how logical models handle schema flexibility. Graph models are designed for domains where interconnection and relationships have equal importance to the entities themselves, making them well-suited to heterogeneous data. Relational models require fixed schemas known in advance, enabling consistency but limiting extensibility to new attributes or entity types. Document models permit irregular, implicit structures where data is self-describing through embedded schema information. For forensic data heterogeneity, graph models are best suited (score 3), followed by document models (score 2), while relational models have the lowest score (score 0) due to schema rigidity.

Cross-source integration (C2) The logical model's relationship semantics directly determine integration capability. Graph models treat relationships as first-class citizens alongside entities, enabling direct representation of connections between forensic artifacts [42, 43]. This structural property allows queries to navigate relationships as native operations. Relational and document models can represent relationships with nested structures, requiring additional application logic to discover and traverse relationships [43]. Graph models, combined with semantic web technologies, can support inference to derive implicit relationships and can naturally reconcile identities through semantic IRIs [42]. Relational models require explicit join operations, while document models require application-level logic to discover relationships. Graph models score 3, while relational and document models both score 1.

Query expressiveness (C3) The logical model's expressiveness for forensic triage queries (5Ws and How) depends on its ability to represent and traverse relationships. Relational logic requires a join operation for each step in a relationship chain, which makes multi-hop traversals difficult to express. However, with primary and foreign keys, these operations are very fast and aggregation is strongly supported. Document models require application-level logic to resolve inter-document references, and deeply nested structures can complicate querying and updating specific sub-elements. Graph logic makes relationship traversal a native operation: finding all nodes reachable within N hops, discovering the shortest paths, and identifying connected components are expressed directly. In contrast, graph models are primarily designed for relationship traversal rather than tabular aggregation. This results in graph models scoring 2, document models scoring 0, and relational models scoring 2 (due to join expressiveness for pairwise relationships) [43].

Provenance and traceability (C4) Provenance (origin of the data) must be represented explicitly in the logical model. Graph models can natively represent provenance as relationships between entities and their sources. Relational models require additional normalized tables and foreign keys to link provenance to entities. Document models can embed provenance as nested structures. All three can support provenance, but graph models integrate it as a first-class relationship type. Graph models score a 3, while both relational and document models score a 2.

Semantic interoperability (C5) Semantic interoperability requires a logical model that represents shared meaning across tools. Ontologies and controlled vocabularies can be used to define graph models by node types, relationship types, and properties. Their flexible schema tolerates incomplete and inaccurate data through optional attributes and sparse fields. In contrast, relational models impose fixed schemas, necessitating alter-table operations to add new entity types or relationships. Document models provide flexibility, but they lack intrinsic semantic structure. Graph models allow new tool outputs to be mapped to existing ontology concepts without modifying the core model. Based on these results, the scores are 3 for graph, 2 for document, and 1 for relational.

Table 5.2 summarizes the previously discussed five weighted criteria (C1–C5) on a scale from 0 to 3, where each raw score is multiplied by the criterion's weight to yield a weighted contribution. The graph model achieves a weighted total of 2.83, outperforming the document model (1.55) and the relational model (1.12) on every criterion. This consistent dominance, particularly on the two highest-weighted criteria C2 (0.28) and C1 (0.22), confirms the graph model as the most suitable logical architecture for the system.

Table 5.2.: Logical model evaluation (0–3 scale)

Criterion (weight)	**Relational**	**Document**	**Graph**
C1 (0.22)	0 (0.00)	2 (0.44)	3 (0.66)
C2 (0.28)	1 (0.28)	1 (0.28)	3 (0.84)
C3 (0.17)	2 (0.34)	0 (0.00)	2 (0.34)
C4 (0.17)	2 (0.34)	2 (0.34)	3 (0.51)
C5 (0.16)	1 (0.16)	2 (0.32)	3 (0.48)
Weighted total	**1.12**	**1.55**	**2.83**

5.1.4. Sensitivity analysis

The evaluation system used is subject to the weight assigned to the criterion. The importance of verifying robustness to weight changes in multi-criteria decision-making is well established

in the literature [40].

We want to confirm that the score of graph models is not only due to the selected weights. To do so, a perturbation analysis is conducted. Each criterion weight w_k is individually shifted by $\delta \in \{-0.10, +0.10\}$. The remaining four weights are rescaled proportionally so that $\sum_{k=1}^{5} w_k = 1$ is preserved. For each perturbed weight vector the weighted total $S_m = \sum_k w_k \, r_{m,k}$ is recomputed for every model $m \in \{\text{Relational, Document, Graph}\}$.

The scope of the sensitivity analysis in this master thesis is intentionally constrained. A more exhaustive study could evaluate alternative scoring methods and weighting schemes, as demonstrated in [40] for engineering decision problems.

Perturbation procedure Let $\mathbf{w} = (0.22, 0.28, 0.17, 0.17, 0.16)$ denote the baseline weight vector. For criterion k the perturbed vector is

$$w_j^{(k,\delta)} = \begin{cases} w_k + \delta & j = k, \\ w_j \dfrac{1 - (w_k + \delta)}{1 - w_k} & j \neq k, \end{cases} \qquad \delta \in \{-0.10, +0.10\}. \tag{5.1}$$

The scores (see Table 5.2) remain fixed, meaning that only the weights change.

Results Table 5.3 reports the weighted totals for all ten perturbation scenarios. In every scenario, the graph model retains the highest score. Its margin over the second ranked model remains over 0.90. The relational model ranks last in all scenarios.

Table 5.3.: Sensitivity analysis: weighted totals under ± 0.10 weight perturbations

Scenario	Perturbed weight	Relational	Document	Graph
Baseline	—	1.12	1.55	2.83
C1 -0.10	0.12	1.22	1.63	2.78
C1 $+0.10$	0.32	1.02	1.47	2.88
C2 -0.10	0.18	1.23	1.66	2.55
C2 $+0.10$	0.38	1.01	1.44	3.11
C3 -0.10	0.07	1.02	1.57	2.76
C3 $+0.10$	0.27	1.22	1.53	2.90
C4 -0.10	0.07	0.98	1.47	2.73
C4 $+0.10$	0.27	1.26	1.63	2.93
C5 -0.10	0.06	1.05	1.41	2.78
C5 $+0.10$	0.26	1.19	1.69	2.88

Break even analysis A break even test identifies the minimum weight $w_k^{\min}$ at which the graph model would not be the model with the highest score. We search for a weight that equalizes the total of the graph with another model. The document model is the model that most easily equals the graph model. Setting $S_{\text{Graph}} = S_{\text{Document}}$ and solving for w_k (while rescaling the remaining weights) yields the values reported in Table 5.4. In every case, the required weight is either negative or exceeds 1, confirming that no single-criterion reweighting can overturn the graph model dominance.

Table 5.4.: Break-even weights: value of w_k at which Graph = Document

Criterion	**Baseline** w_k	**Break-even** w_k
C1	0.22	> 1.00
C2	0.28	< 0.00
C3	0.17	> 1.00
C4	0.17	> 1.00
C5	0.16	> 1.00

Conclusion The sensitivity analysis demonstrates that the selection of the graph model is robust to weight uncertainty. The ranking is preserved across all ten ± 0.10 perturbation scenarios, and the break-even analysis shows that no feasible single-criterion weight assignment can elevate the document or relational model above the graph model. The recommendation is therefore insensitive to plausible variations in the importance assigned to individual criteria.

5.1.5. Justification of the selected approach

Table 5.2 shows that the graph logical model achieves the highest score (2.83) across all criteria, with particular strength in cross-source integration (C2), query expressiveness (C3), and semantic interoperability (C5). The sensitivity analysis performed confirms that the model selection is not biased by the criterion weights.

The advantage of the graph model is its conceptual alignment with the forensic problem. Forensic analysis is fundamentally about discovering relationships between artifacts and events. By treating relationships as first-class entities, a graph logical model directly represents this core task. The model natively supports the triage queries defined in the requirements (5Ws and How) without requiring application logic.

Relational and document models can technically store forensic data, but they require workarounds: relational models might demand schema modifications for each new forensic

tool, while document models provide flexibility without inherent semantic structure. In contrast, the graph model separates concerns, distinguishing the logical representation (entities, relationships, and semantics) from the physical implementation. This distinction is essential for a tool-agnostic, extensible model.

To be more precise, the proposed unified model is specified as an ontology that formalizes domain concepts, relationships, and constraints. It constitutes the semantic backbone of the knowledge graph, which instantiates these ontology classes and properties with case-specific forensic data. The mapping layer transforms heterogeneous tool outputs into instances of the ontology, while the query interface allows analysts to explore the knowledge graph using the defined semantics.

5.2. Architecture

The architecture proposed in this work follows the Abstract Reference Architecture (ARA) [44] and is composed of three layers.

- **The unified data model:** An ontology that defines core forensic concepts (Process, File, etc) and their relationships. This model should respect the requirements previously defined. By defining a unified data model, the design directly addresses the Ontology Development task within the Knowledge Acquisition and Integration layer of the ARA.
- **Mapping layer:** A set of transformation rules that take heterogeneous tool outputs and map them to instances of the previously defined ontology, resolving schema differences and normalizing terminology. The objective is to generate a knowledge graph connecting entities associated with each other across various tools. This mapping needs to be done using the ontology as a semantic framework to guarantee uniform representation of data. This layer fulfills the ARA's requirement for Data Lifting, as it transforms heterogeneous resources into linked entities that serve as the "semantic backbone" of the system.
- **Query and visualization interface:** This layer allows to query the knowledge graph. The types of queries are questions related to triage, including visualization. It must be able to handle triage related queries such as who, what, when, where, and how, and use visualizations such as graphs or timelines. This interface aligns with the Knowledge Consumption layer. Note that [44] presents the Knowledge Consumption Layer with generic views that tend to be able to answer any queries, whereas this implementation is oriented towards triage questions.

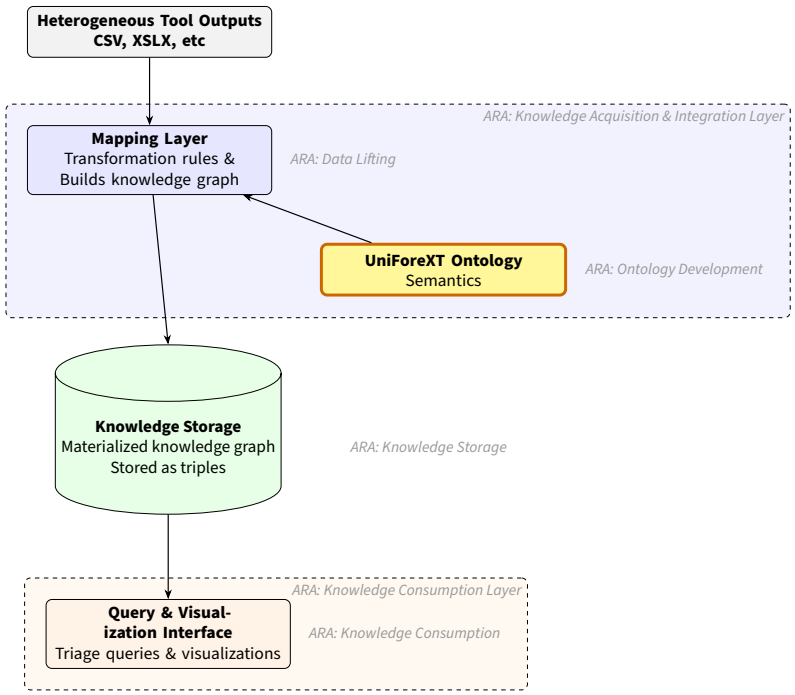


Figure 5.1.: System architecture with the UniForeXT ontology highlighted

The architecture of this work is represented in Figure 5.1. The two latter components will be detailed in the next chapters. The rest of this chapter focuses on the design of the unified data model itself, highlighted in the figure.

5.3. Ontology Modeling

We divide the modeling of the unified data model as an ontology into three steps. First, the competency questions that the ontology should address are identified. Second, the core concepts and relationships are defined. The competency questions and the core concepts and relationships are based on the analysis of forensic tool outputs and the previously defined requirements. Third, the ontology is implemented using a standard format.

These steps follow the ontology engineering methodology introduced by SABiO [45], a popular methodology for developing ontologies in a structured manner. The methodology stresses the importance of posing appropriate competency questions during the development of the ontology, as well as the iterative nature of the development process in order to ensure that the ontology meets its intended purpose.

5.3.1. Competency questions

Competency Questions (CQs) are natural language questions. From these questions, patterns of questions that the knowledge system (in this case an ontology) should be able to answer are extracted. The notion of competency questions was introduced by [46], where competency questions are described as an informal and structured approach to defining requirements for an ontology to answer. Failure to provide meaningful answers to the questions addressed by the ontology implies that the ontology does not fulfill its intended function. Competency questions enable the author to decide the breadth and depth of an ontology, as well as which concepts, relations, and properties are essential to include in the ontology. Moreover, since competency questions are stated in natural language, they tend to emphasize domain knowledge over formal logic, making them easier for designers who lack knowledge about ontology representation languages [47].

From a methodological standpoint, CQs can be compared to test cases in test-driven software engineering. In essence, they serve as specifications for an ontology to be implemented and are employed in assessing whether an ontology is built correctly. CQs are regarded as answerable when providing a meaningful output. Consequently, a CQ imposes requirements on the presence of relevant classes, properties, and relations within an ontology. The meaningful answerability of CQs is strongly associated with the idea of presupposition in linguistics [47]. Each CQ is based on the assumption that certain classes and relations must be present in an ontology for a question to be meaningful. For example, the question "Which processes implement an algorithm?" presumes the existence of such classes as `Process` and `Algorithm`, as well as the property that connects them. Moreover, it is implicitly assumed that processes implementing algorithms exist and coexist with processes that fail to implement algorithms. However, if these ontological concepts and relations cannot be captured, then it is impossible to provide an answer to the question, indicating a gap in the ontology design. This highlights the importance of having semantic knowledge within the ontology. Through the analysis of questions and their assumptions, ontology engineers are able to develop such a model [47].

A recent article provides an evaluation of LLMs for generating CQs [48]. One of its authors is the creator of the NORIA-O ontology discussed previously in Chapter 4. The article limits its scope to evaluating LLMs' ability to generate CQs for existing ontologies using multiple types of prompts to input the ontology into the LLM for evaluation. According to their empirical study, LLMs demonstrate a modest yet promising ability to generate competency questions for ontologies. Precision scores are generally low across models and ontologies, though performance improves meaningfully when few-shot prompting (prompting engineering technique consisting of providing some examples, or "shots", to guide the LLM towards more

precise, structured, and pertinent answers [49]) is used and when property information is included in prompts. Importantly, low scores don't necessarily mean the generated CQs are useless, because they may represent novel, valid questions not captured by the creator of the ontology.

The Table A.3 (in Appendix A) lists the competency questions that the unified data model should be able to answer. These questions are derived from the requirements and represent typical questions that the ontology should be able to answer. They cover core triage questions (5Ws and How), cross-tool integration, attack semantics, temporal relationships, provenance, and data quality. These questions will guide the design of the ontology, ensuring that it provides sufficient expressiveness for forensic triage. These questions will also depend on the nature of the data being modeled.

The type, arity, and archetypes of each CQ are also indicated. They are features, as defined in [47], that can be used to analyze the structure of CQs and to guide the modeling of the ontology. The type determines whether the question is a selection (retrieving nodes), a counting (number of nodes), or a binary. The arity tells whether the predicate is unary (single entity property) or binary (relation between two entities). Such an analysis will help us ensure that the ontology contains all the necessary classes and properties needed to answer such queries. The archetypes are defined by [47] and represent common patterns in CQs that can be used to guide the modeling of the ontology. For example, archetype 1 corresponds to questions that involve finding entities that meet specific criteria, whereas archetype 8 relates to problems that involve finding sequences or relationships between entities. By identifying the archetypes of the CQs, we can ensure that the ontology is designed to support these common query patterns. Table A.1 provides a list of the archetypes defined by [47].

After defining the CQ archetype of a CQ, it is possible to identify the required classes, object properties, and data properties by looking at the archetypes table (see Table A.1). The arity enables the identification of the types of properties that the CQs require. Binary CQs require object properties and unary CQs require data properties.

5.3.2. Conceptualization and design

In the conceptualization phase, the domain was modeled by identifying the core forensic concepts and the relationships between them from the competency questions, requirements analysis, and the observed tool outputs.

The conceptual model was organized around a small set of top-level categories. At the highest level, the ontology distinguishes generic domain entities from value-like concepts. On the entity side, the model includes forensic artifacts and observable elements such as

ufr:Process, ufr:LogFile, ufr:Session, ufr:Event, ufr:Detection, ufr:Computer, and ufr:User. On the value side, it includes controlled concepts such as ufr:Path, ufr:LogonType, ufr:AttackTactic, and ufr:AttackTechnique. This structure allows the ontology to separate concrete forensic objects from descriptive or classificatory values.

Relationships between concepts were also defined at this stage. For example, ufr:hasProcess, ufr:hasUser, ufr:hasComputer, ufr:hasLogFile, ufr:hasDetection, ufr:hasAttackTactic, and ufr:hasAttackTechnique capture the main links between artifacts, events, detections, and controlled vocabularies. Data properties such as ufr:hasTimestamp, ufr:hasEventID, ufr:hasName, and ufr:hasValue were used to represent literal attributes. This conceptualization forms the semantic foundation of the unified forensic data model. It supports the mapping of heterogeneous tool outputs into a consistent knowledge graph.

Modeling decisions and rationale First, a central decision was to distinguish Independent_entity from Value. Concepts such as ufr:Process, ufr:Event, ufr:Detection, or ufr:User represent concrete forensic objects or observations, while concepts such as ufr:AttackTactic, ufr:AttackTechnique, and ufr:LogonType represent controlled classificatory values. This separation avoids mixing observed evidence with taxonomy-like concepts and makes mappings from heterogeneous tools more stable.

Second, for ATT&CK and logon semantics, the model favors explicit semantic links (e.g., ufr:hasAttackTactic, ufr:hasAttackTechnique, and ufr:hasLogonType) instead of only storing plain text literals. This supports normalization across tools and enables reasoning and filtering by shared concepts rather than string matching.

Third, process relationships were modeled explicitly (ufr:hasProcess, ufr:hasParentProcess, ufr:hasChildrenProcess) rather than being flattened into attributes. The rationale is that forensic triage frequently depends on lineage and propagation paths, which are naturally represented as graph relations and are easier to traverse in multi-hop investigations.

Fourth, several properties were constrained with functional/cardinality restrictions (e.g., one ufr:hasComputer or one ufr:hasUser in specific contexts). The intent is not to force complete data but to avoid contradictory assertions while keeping the model compatible with partial evidence.

Finally, inverse properties were introduced systematically (for example ufr:hasDetection/ufr:isDetectionOf, ufr:hasProcess/ufr:isProcessOf) to improve bidirectional navigation and to make analyst queries less dependent on a single traversal direction. This

was also aligned with the ontology quality recommendations that were identified during the validation process.

This ontology organizes concepts into four semantic families: the event family, the artifact family, the detection family, and the controlled value family. This structure improves the model's scalability and supports consistency when incorporating other forensic tools or logs.

5.3.3. Implementation

The ontology is encoded in OWL/RDF using Protégé [50], but deliberately restricts itself to lightweight RDFS-level semantics, with stricter constraints enforced separately through SHACL. Protégé is a widely used ontology editor that provides a graphical interface for defining classes, properties, and relationships. The SABiO methodology recommends using OntoUML [45].

The Resource Description Framework Schema (RDFS) is used to define simple vocabularies and lightweight ontologies. It operates on top of the Resource Description Framework (RDF). RDFS provides constructs for defining classes, properties, and relationships. This allows domain knowledge to be represented in a structured, machine-readable format. RDFS defines a set of meta-properties used to define assumptions in ontologies, such as `type` for instance-of relationship, `rdfs:subClassOf` for class hierarchies, `rdfs:domain` and `rdfs:range` for property constraints, and `rdfs:label` and `rdfs:comment` for annotations [51]. In terms of performance, RDFS allows for more efficient querying because it is less expressive than OWL. Its simplicity also makes it easier to implement and maintain. This is important for a model that needs to be extensible and adaptable to new forensic tools and data sources.

The simplicity of RDFS also causes multiple limitations that are mostly related to its low expressiveness. Being less expressive compared to other ontology languages such as OWL, RDFS cannot use some vocabulary needed to describe complicated semantics within a domain and implement sophisticated restrictions [52, 53].

First, RDFS cannot define cardinality since the language does not allow specifying the exact minimum or maximum cardinalities of properties. In other words, it is impossible to state how many times a certain property can be used to describe a particular resource [52, 54]. Nevertheless, cardinality can be tested in SHACL.

Second, RDFS cannot declare properties to be transitive, symmetric, or functional, nor can it identify the inverse of a property [52, 54].

Additionally, RDFS cannot define class combinations such as conjunctions, unions, or disjunctions. It also cannot declare that two classes are disjoint or define a class by enumerating its instances [52, 53, 54].

5.3.4. Resulting ontology

Figure 5.2 presents a high-level conceptual view of the UniForeXT ontology. It was generated using WebVOWL [55]. The specification of the ontology can be accessed using the following link `https://louanrobert.github.io/UniForeXT/`. It was automatically generated using WIDOCO [56]. The ontology is available on the same page or inside the `ontology` directory of the project repository [57, 58].

5.4. Evaluation

Using OOPS! [59], a tool for evaluating ontologies against common pitfalls and best practices, the unified data model ontology was analyzed to identify potential issues in its design and implementation to respect the **NFR6**. The evaluation revealed several areas for improvement, such as missing annotations for some classes, missing domain and range definitions for properties, some inverse relationships that could be defined, and the usage of some recursive class definitions. These issues were addressed iteratively to enhance the ontology's quality, consistency, and usability. The final version of the ontology was then re-evaluated with OOPS! to confirm that the identified issues were resolved and that the ontology adheres to best practices for design and implementation.

Figure 5.3.: OOPS! Important pitfalls [59]

The model presented in this chapter reflects the improvements made based on the OOPS! evaluation, ensuring a robust and well-structured ontology for representing unified forensic data.

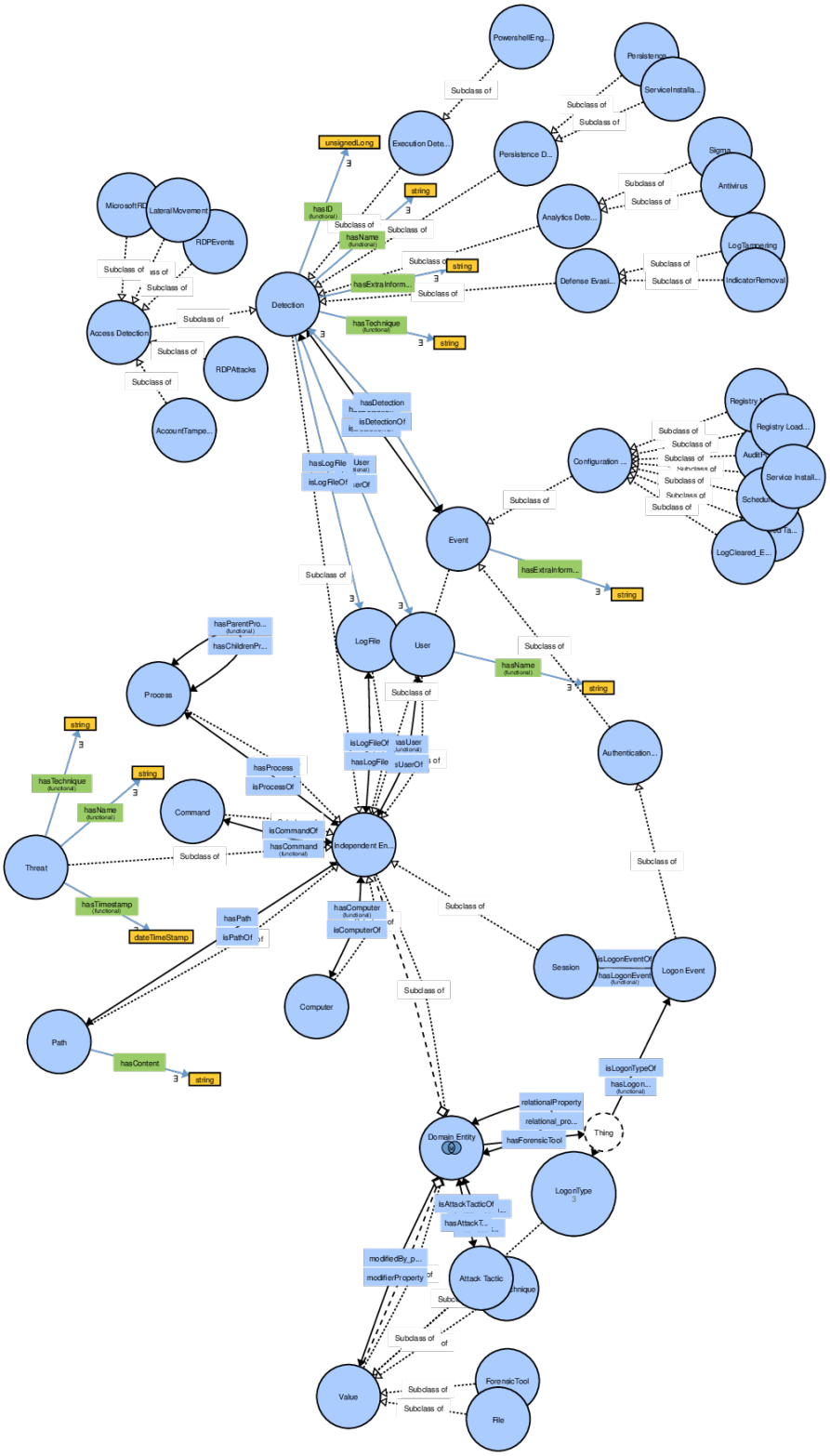


Figure 5.2.: High-level conceptual view of UniForeXT

5.5. Validation

To complement the evaluation, this section validates the ontology against competency questions (CQs) using authoring tests derived from the methodology proposed in [47].

To validate the ontology, [47] used Authoring Tests (ATs). ATs employ presuppositions, which are conditions that must be met for a question to have a meaningful answer.

The ATs are based on the previously defined CQs (see Table A.3) and their features (type, arity, and archetypes). For each CQ, presuppositions are extracted. With the archetypes, we know how to extract the presuppositions of the question using both Table A.1 and Table A.2.

Let's take the well-known pizza example. We have the question "Which pizzas contain pork?" that maps to the archetype 1 "Which [CE1] [OPE] [CE2]?". To identify the presuppositions, consider that each feature of the question implies presuppositions. For this question, *Pizza*, *Pork*, and *contains* must exist in the ontology vocabulary. Pizza $\sqcap \exists$ contains.Pork must be satisfiable (some pizza can contain pork). Pizza $\sqcap \forall$ contains.$\neg$Pork must also be satisfiable (some pizza can not contain pork, otherwise the filter is trivial).

Following [47], each AT type targets a specific modeling template and therefore matters for different reasons:

- **OCC** Occurrence checks that the necessary domain classes and properties are present (vocabulary coverage)
- **SAT** Class satisfiability ensures that a class can actually have instances rather than being logically inconsistent
- **REL** Relation satisfiability includes both positive and negative existential forms so that relations can both hold and not hold (avoiding trivially empty or universally true answers)
- **META** Meta-instance verifies that properties are declared with the correct meta-type (object vs datatype)
- **CARD** Cardinality tests ensure that numeric constraints are meaningful and do not render queries impossible
- **RNG** Range validates that property values fall within the expected data types or classes so answers are comparable

Together, these tests operationalize the presuppositions behind competency questions and help produce cooperative, discriminative answers rather than trivial or misleading results.

The AT type assigned to each CQ depends on the presupposition it is meant to stress: OCC occurrence and SAT satisfiability are sufficient for binary questions such as “Are there any...” because the main risk is missing vocabulary or an inconsistent class, while REL relation pairs are added when the question asks for a link between two concepts and the ontology must support both the presence and absence of that link to avoid trivially empty or universally true answers. META Meta-instance tests check whether the modeling choice itself is typed correctly, CARD cardinality tests are used only when the CQ depends on counts or uniqueness constraints, and RNG range tests are reserved for questions whose answers depend on a datatype or target class being well formed. In that sense, the selection is not arbitrary: it mirrors the minimal set of presuppositions needed for a question to be answerable.

Table A.4 contains all the authoring tests based on the competency questions defined in Table A.3 in Appendix A.

These ATs checks were applied manually to the ontology to verify its validity. This process led to several minor modifications in order to satisfy all the tests. The fact that changes were required highlights the relevance of this testing methodology. Although ATs are defined strictly as tests by [47], the author argues, based on the use of the proposed methodology, that ATs could serve a broader purpose: not only as validation mechanisms but also as guidance for ontology construction and refinement throughout the development process.

5.6. Data Validation

The suggested model’s data must be consistent and integrity must be maintained. This will require the definition of a set of rules and constraints. These rules would ensure that the data corresponds to the expected format, thereby ensuring the consistency of the knowledge graph.

These rules are formulated using the Shapes Constraint Language (SHACL). SHACL is a W3C recommendation used to validate RDF graphs based on a specific set of constraints. SHACL provides facilities to formulate SHACL shapes, which define constraints regarding cardinality, types, ranges of values, format, etc [60]. Utilizing the power of SHACL rules on our knowledge graph allows us to ensure that the data satisfies the structural and semantic requirements posed by the ontology. For instance, using SHACL shapes, we can force each `ufr:Detection` node to have a valid `ufr:hasTimestamp` property in date-time format, or to ensure that each `ufr:Process` node possesses at least one valid `ufr:hasUser` property, indicating an existing relationship between the `ufr:Process` node and the `ufr:User` node. These SHACL shapes ensure the integrity and consistency of the data stored in the knowledge graph. Each

SHACL shape indicates the target classes of nodes along with constraints that need to be satisfied [60].

These constraints will be defined based on the competency questions and the expected data structure of the knowledge graph. The implementation of the constraints will have sufficient flexibility due to the nature of forensic data which may not always be complete or accurate.

The implementation of the shapes is done using the SHACL syntax. The SHACL shapes are defined in a separate file, which is called a shapes graph [60]. During the data ingestion process, this shapes graph is applied to the knowledge graph. Its function is to validate the data before it is saved on the disk. Any violation of the constraints will be logged.

At the most general level, the shapes graph distinguishes between domain entities and the more specific categories used to represent forensic observations and supporting evidence. Independent entities are treated as controlled records that may carry a limited and well-typed set of descriptors. They may have at most one name, description, timestamp, technique, computer reference, user reference, and command reference. The data properties are restricted to the appropriate data types, while the object properties must point to the corresponding ontology classes.

The central observational concepts are detections, events, and threats. A detection must be linked to at least one log file and may additionally be associated with a user, a computer, an identifier, a name, a technique, a description, a timestamp, and a severity level. The level is constrained to a closed vocabulary consisting of `High`, `Info`, `Low`, and `Medium`. An event is similarly grounded in at least one detection and may also reference a computer, log file, user, free-text contextual information, an event identifier, and a timestamp. A threat is modelled as a temporally anchored concept that may be associated with a command, computer, process, and user, but must have exactly one name, one technique label, and one timestamp. This gives threats a canonical identification pattern and ensures that their description remains internally coherent.

The supporting forensic entities are constrained in a similarly disciplined manner. A process may reference at most one command, computer, parent process, and name, thereby reflecting the idea of a single execution context with limited provenance information. A log file can be associated with one path only, whereas one path must have at least one value of textual contents. One session can optionally include one logon event and one user, and one logon event can reference one logon type only. A computer is described through optional but functionally limited domain, hostname, and IP values. A user, by contrast, must always have exactly one name. The registry-modification configuration event is also constrained by a single decimal-valued attribute, which suggests a narrowly typed quantitative parameter.

In addition to these class constraints, SHACL explicitly validates the intended usage of the main object properties by enforcing their domain and range. For example, properties such as `ufr:hasUser`, `ufr:hasComputer`, `ufr:hasDetection`, `ufr:hasLogFile`, `ufr:hasProcess`, and `ufr:hasPath` are restricted so that only the expected source and target classes may participate in them. These constraints are important because they make the ontology semantics executable. They do not merely document the model, but verify that the RDF data respects the intended typing of relationships.

A disjointness-style restriction states that independent entities cannot also be values. This is a global logical constraint rather than a local property rule, and it prevents overlap between objects meant to denote concrete forensic entities and those meant to represent value concepts. Overall, the SHACL graph turns the ontology into a precise validation framework that enforces structural consistency across the forensic data model.

Furthermore, the article [61] provides a framework for the generation of SHACL shapes from an existing ontology. The framework is composed of Astrea-KG, it consists of a set of mappings that represent the corresponding conceptual equivalents between SHACL and ontology constraint patterns, and Astrea, which is a tool that takes Astrea-KG mappings and an ontology as input to generate a SHACL shapes graph. This framework was used to generate SHACL shapes from the defined ontology. These rules were compared with the already existing shapes.

Astrea generated 17 equivalent shapes to the ones already defined and generated an impressive total of 257 additional shapes that were not previously defined (77 shapes were defined earlier). This large number of additional shapes is due to the fact that Astrea generates shapes for all the classes and properties defined in the ontology, while the previously defined shapes were focused on the main concepts and relationships. Additionally, Astrea tends to be more precise in date and time validation. All of the new shapes were tested on a sample of data to check for consistency and to ensure that they do not introduce unintended constraints. Astrea was intended to be a complementary tool "to try" but it turned out to be a useful tool for identifying potential constraints that were not previously defined and ensuring that the ontology is well-constrained and consistent.

Figure 5.4 is a UML diagram displaying the SHACL shapes.

5.7. Summary

This chapter presents UniForeXT: a data model used to unify heterogeneous outputs from digital forensic tools, making artifacts, events, and entities semantically coherent, queryable,

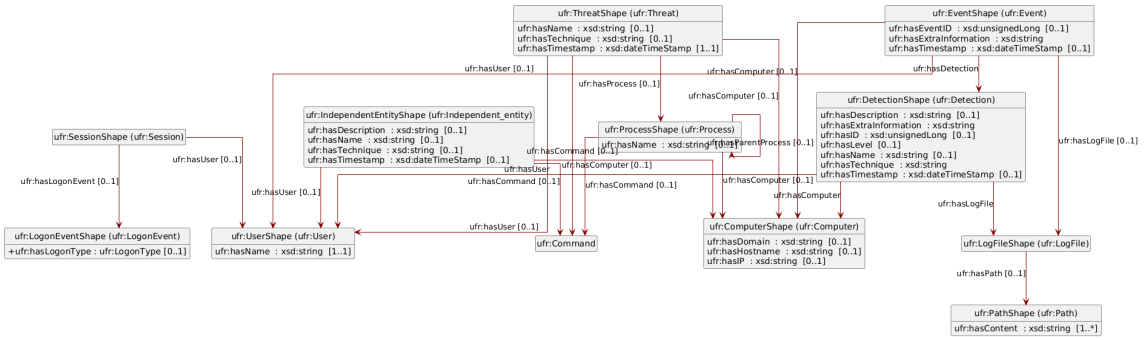


Figure 5.4.: UML diagram of SHACL shapes, generated with the help of SHACL Play! [62]

and traceable across tools.

Three candidate logical models were evaluated (relational, document, and graph) against five weighted criteria: semantic integration, cross-source integration, query expressiveness, provenance/traceability, and semantic interoperability. The graph model scored highest, primarily because it treats relationships as first-class citizens, which aligns naturally with forensic analysis, where discovering connections between artifacts is the core task.

The architecture of the system is divided into three layers: the ontology (core forensic concepts and relationships), a mapping layer (transforming tool outputs into knowledge graphs conforming to the ontology), and a query/visualization interface (supporting 5Ws & H triage queries).

A set of structured natural language questions (competency questions) was defined to scope and validate the ontology. These cover core triage queries, cross-tool integration, attack semantics (MITRE ATT&CK), temporal/lineage analysis, provenance, and data quality.

The ontology was conceptualized and implemented in OWL using Protégé. Key modeling decisions include separating concrete forensic entities (e.g., `ufr:Process`, `ufr:Detection`, `ufr:User`) from classificatory values (e.g., `ufr:AttackTactic`, `ufr:LogonType`), explicit process lineage relationships, and bidirectional inverse properties. Validation was performed using the OOPS! tool to catch common ontology pitfalls.

SHACL shapes were defined to enforce structural constraints on the knowledge graph (cardinality rules, value types, property domains/ranges, and vocabulary restrictions). The Astrea framework was also used to auto-generate shapes from the ontology, producing 257 additional constraints beyond the 77 manually defined, proving a useful complement for thoroughness.

Chapter 6 Mapping

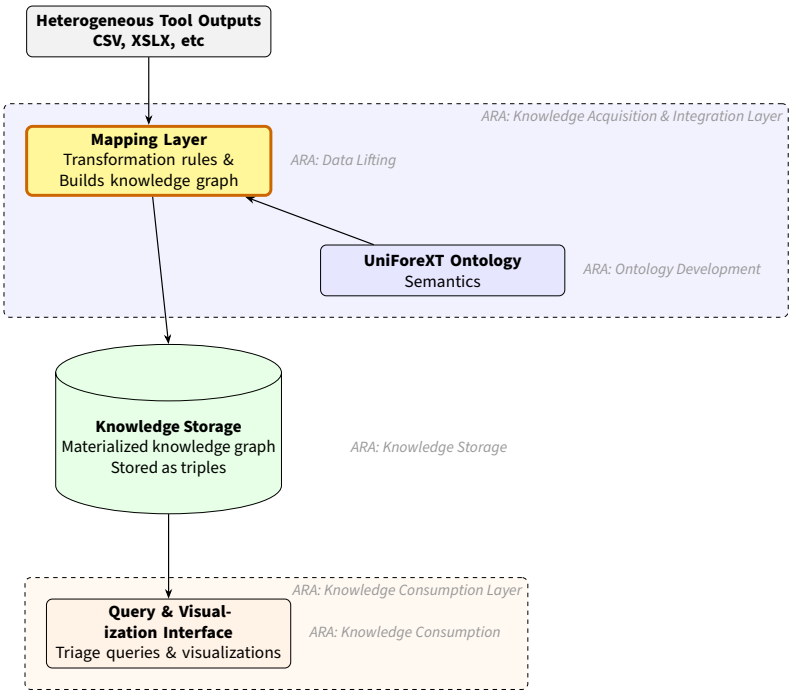


Figure 6.1.: System architecture with the mapping layer highlighted

The role of the mapping layer is to serve as the bridge between the source data and the knowledge graph. This is highlighted in Figure 6.1. The mapping layer's task is not only to translate rows of information from various files into RDF triples, but also to translate each piece of data according to the rules provided in the ontology. This ensures that the resulting triples are meaningful in the knowledge graph.

The source code of the mapping layer is publicly available in the `semantic-mapper` directory of the project repository [57] and has been archived on Zenodo for long-term preservation [58].

The architecture of the mapping layer is presented in Figure 6.2. It comprises six principal components arranged in a vertical pipeline. First, the ontology and the mappers configuration are loaded. Next, the pipeline runs the ingestion stage. This stage reads the output of the forensic tool, maps it according to the mapper configuration and creates RDF triples. This stage runs for each ingested file. Next, the resulting graph is validated against the SHACL shapes. Finally, the triples are saved to disk.

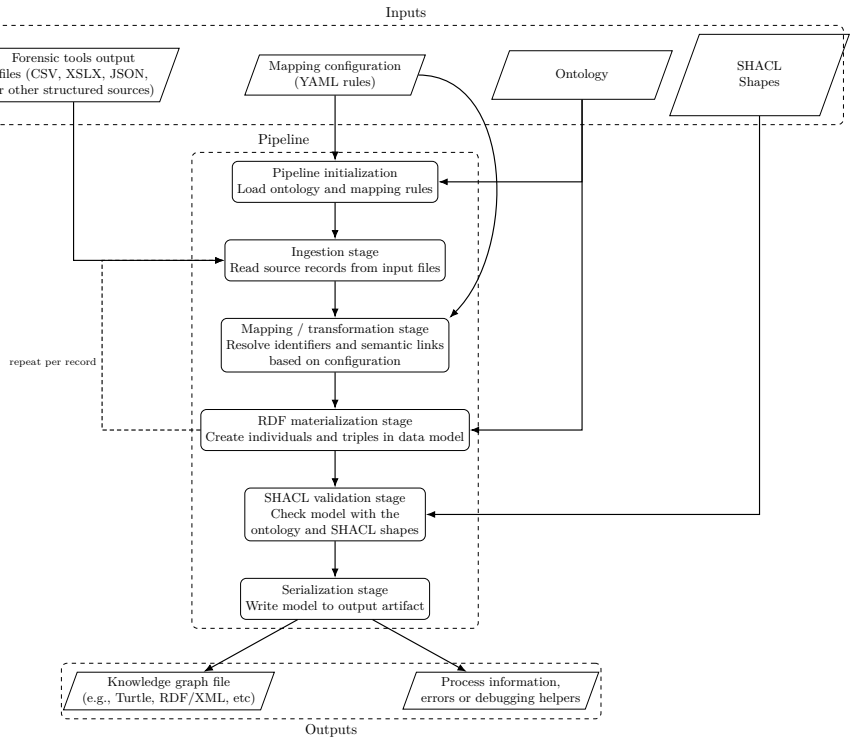


Figure 6.2.: Mapping layer architecture

6.1. Programming Language

The programming language was selected based on the following criteria:

- Heterogeneity of input formats
- Strong library support for RDF and OWL
- The need for flexible, maintainable mapping configurations
- Efficient processing of large datasets
- The ability to run on diverse platforms (the mapping layer will run inside a cloud pipeline)

From these criteria, Java was the natural choice for implementing the mapping layer. Java has mature libraries for RDF and OWL, such as Apache Jena [63], which provide robust support for ontology manipulation and graph construction. Java has strong typing and object-oriented features facilitating the development of a complex mapping system. Regarding Java support for parsing different input formats, it has extensive libraries that can be obtained either from the standard libraries or from third-party libraries.

Scala would have been a good alternative because it runs on the JVM, leverages Java libraries, and provides more concise syntax and functional programming features. A panoply of languages would have been suitable for this work.

6.2. Mapping Process

At first, the initial design of the mapping process was to create a separate Java class for each source file, with hardcoded logic to parse the file and generate the corresponding RDF triples. However, this approach quickly proved to be unmaintainable and unflexible, especially as the number of source files is large and evolving. Each new source would require a new Java class and any change to the file format would require code changes, which is not ideal for a system that needs to adapt to new forensic tools and future changes in existing tools.

To address this challenge, the design was shifted towards a more configuration driven approach. Instead of hardcoding the mapping logic in Java classes, a YAML configuration system was introduced to define the mapping process for each source file. This configuration specifies how to extract identifiers, which class to instantiate, which properties to populate, and how to link individuals together. The Java code then becomes a generic engine that reads the YAML configuration and applies it to each source record, allowing for a much more flexible and maintainable mapping process. This design allows new sources to be added simply by creating new YAML configurations without changing the underlying Java code, and it also makes it easier to adapt to changes in existing file formats by updating the YAML configuration rather than modifying the code.

Furthermore, this design makes it easy to ingest new file formats in the future. For example, if a new forensic tool produces JSON output, a new implementation of the file ingestion interface can be created to parse JSON and add the new source type to the mapping system. This can be done without affecting the existing ingestion logic. Separating the parsing of input files from the semantic mapping to the ontology enhances the system's extensibility and maintainability.

This design provides a clear ingestion configuration, allowing for straightforward mapping of source files. The YAML configuration is human-readable and can be modified by users who are familiar with YAML syntax, without requiring an understanding of the underlying source code.

6.2.1. Positioning with respect to knowledge graph construction frameworks

To perform the mapping, an alternative approach would have been to rely on declarative Knowledge Graph Construction frameworks such as RML and YARRRML. These frameworks are designed for transforming heterogeneous data sources into RDF graphs [64]. The KGC frameworks were not identified during the initial design stage of the project. Therefore, a custom YAML-driven semantic mapping pipeline was developed. The identification of KGC frameworks came up during further reviews and consultations with Prof. C. Debruyne.

Interestingly, the resulting architecture converged on principles similar to those used in existing knowledge graph construction systems, such as declarative mappings (specifically YARRRML, which uses YAML for declarations), the separation of parsing and semantic transformation, and schema-driven graph generation. This convergence indicates that the proposed design addresses genuine architectural challenges that arise when integrating diverse forensic artifacts into a knowledge graph.

An interesting feature of this work that is absent from RML and YARRRML is the ability to define generic mappings. This feature eliminates the need to define the same mapping for different source files. Otherwise, maintenance becomes cumbersome. That being said, YARRRML users could leverage YAML anchors, but this technique would require some tinkering due to the inherent nested lists caused by anchors (see Listing B.1).

The existence of mature frameworks, such as RML and YARRRML, does not render the proposed approach invalid. Rather, their presence highlights that the developed prototype adheres to concepts in line with the latest knowledge graph construction practices. Adopting standardized solutions like RML and YARRRML could offer enhanced interoperability, tooling support, long-term maintainability, and integration with the broader Semantic Web ecosystem.

6.2.2. Implementation details

The process begins with the ontology loader. It reads the previously defined ontology from the disk, exposes the ontology model, and creates a knowledge graph for ingested instances. The knowledge graph is initialized with the ontology prefix definitions. This design makes the resulting graph explicit. The schema remains the authoritative conceptual model while the ingested data becomes a conforming instance layer that inherits its semantics from the ontology instead of redefining them.

The ingestion pipeline is responsible for orchestration. It selects the source files to ingest, chooses the appropriate mapper for each file, and delegates the record-by-record transformation to a source ingester. The source ingester reads inputs and passes each record to a mapper that knows how to interpret that specific tool output. This separation is important from a data engineering perspective because it isolates file parsing from semantic transformation: the reader only concerns itself with structure, while the mapper concerns itself with meaning. The source ingester is transparent to the mapping logic, which allows for easy extension to new file formats by simply implementing a new reader without changing the mapping rules. Indeed, to understand each other, the reader and the mapper use a data structure that represents a source record as a map of names to values, which abstracts away the details of file parsing.

The semantic core of the system is the YAML-driven mapper. Each mapper configuration declares the class to instantiate, the identifier strategy to use (concatenation, hashing, etc.), optional static properties, reusable generic mappings, and explicit field-to-property mappings. For every source record, the mapper first constructs a stable identifier from one or more source fields. That identifier is then used to create an individual of the target class, ensuring that the same entity is represented consistently across the graph. This is a critical requirement for cross-source integration. It enables repeated observations of the same forensic object to be grouped together under a single graph node, rather than being scattered across duplicate resources. This natural alignment between the identifiers is a strength of this logical model because it allows the mapper to generate a graph that is not only semantically rich but also naturally structurally coherent.

The mapper then populates that individual with properties. Static properties encode invariant semantic assertions that hold for all instances produced by a mapper (i.e., all instances produced by the mapper `AntivirusMapper` will have a static property "Past antivirus detections" with the property `ufr:hasDescription`). Field mappings translate source fields into individuals, creating the corresponding properties, optionally adding prefixes, preserving multiplicity where required, or enforcing uniqueness when the ontology semantics require a single canonical value. The same care regarding identifiers applies.

Reusable, generic mapping groups provide a second layer of abstraction. Some forensic tools produce multiple output files with partially similar structures. Rather than duplicating common patterns, such as timestamps, users, computers, or log files, across every mapper definition, the YAML configuration defines shared mapping fragments that can be reused by multiple source types. This makes the mapping rules easier to maintain. Repeated forensic concepts are modeled once and reused consistently. These generic mappings are local to the YAML file.

The final step in the mapping process is to validate the generated graph against the defined SHACL graph. This step ensures the mapping rules are implemented correctly and the resulting graph adheres to the structural constraints of the ontology. If any violations are detected, they can be traced back to specific mapping rules or source records. This allows for the iterative refinement of the mapping configuration until a valid graph is produced.

This mapper can be described as a schema-driven semantic transformation layer. Its role is to operationalize the ontology by translating heterogeneous source artifacts into individuals and relations, building a knowledge graph that is both faithful to the source evidence and structured enough to support integration, and analysis. The architectural choice to externalize the mappings in YAML further strengthens this design because it separates the ontology backed semantics from the Java implementation and makes the transformation layer adaptable as new forensic sources or ontology concepts are introduced.

After all the source files have been ingested, the resulting knowledge graph is saved to disk in RDF or Turtle format. This makes the graph available for the next layer of the architecture: the query and visualization layer. This step is highlighted in Figure 6.3.

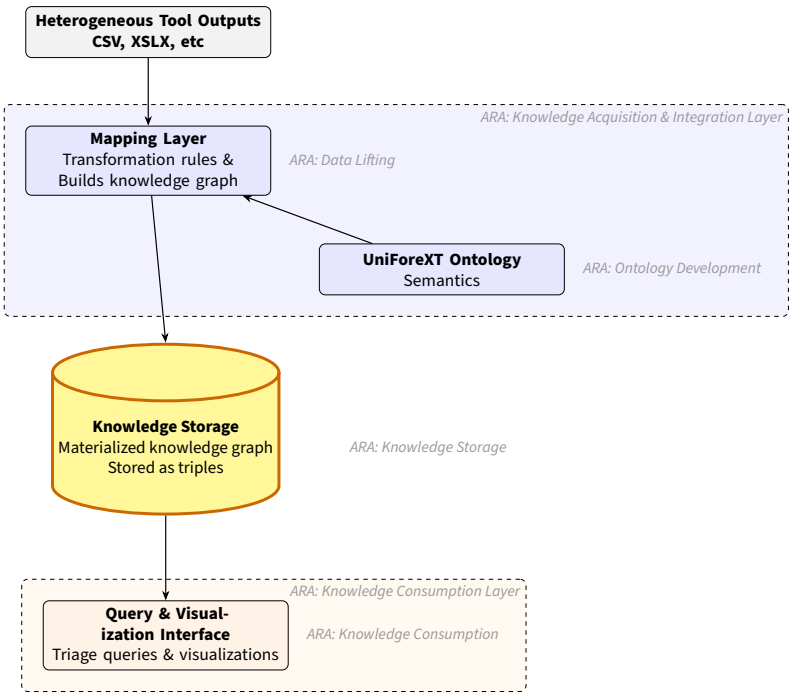


Figure 6.3.: System architecture with the storage layer highlighted

Figure 6.2 illustrates the architecture of the mapping process, showing how the ontology loader, ingestion pipeline, source ingester, and YAML-driven mapper interact to produce a coherent knowledge graph from diverse forensic source files.

Listing 6.1 shows the tool's output. The pipeline stages are clearly visible. First, mappers are loaded from the configuration directory. Then, source files from heterogeneous tools, such as Chainsaw, Evtxhussar, Hayabusa, WinSearchDB, and Zircolite, are ingested sequentially. A progress bar shows the progression of each source file's ingestion. This feature makes it possible to track the progress of big files. Finally, the triples are validated against the SHACL shapes.

```
[LoadMappersStage] Loading mappers from config directory:
↪  ..\ingestion-config
[ExecuteIngestionTasksStage] Ingesting file:
↪  C:\Users\Robert_Louan\Downloads\DFIR_Automation_Results_2\wks02\
↪  QuickWins\Chainsaw_results\antivirus.csv
[ExecuteIngestionTasksStage] Ingesting file:
↪  C:\Users\Robert_Louan\Downloads\DFIR_Automation_Results_2\wks02\
↪  QuickWins\Chainsaw_results\sigma.csv
[...]
[ExecuteIngestionTasksStage] Ingesting file: C:\Users\Robert_Louan\
↪  Downloads\DFIR_Automation_Results_2\wks02\Zircolite_results.csv
```

Listing 6.1: Example output of the mapping tool

6.2.3. Software architecture and object-oriented design

At a high level, the implementation follows a layered structure:

- **Orchestration:** orchestrate the pipeline stages and context
- **Configuration stage:** load YAML mappers and create corresponding objects
- **Task definition stage:** define the different tasks to execute and schedule them from the YAML configuration
- **Ingestion stage:** read the defined file and map it using the corresponding mapper and file format abstractions
- **Validation stage:** use SHACL shapes to validate the knowledge graph
- **Semantic layer:** provide methods for ontology (read-only) and knowledge graph manipulations (triple storage)

This decomposition follows the Single Responsibility Principle [65, 66]. This principle was invented by Robert C. Martin. It is a fundamental design principle in software engineering that states that a class or module should have only one reason to change. It should have only

one responsibility. By adhering to this principle, the codebase becomes more modular, easier to understand and maintain. In the context of the mapping process, this means that changes to the ontology loading logic will not affect the ingestion orchestration or the semantic mapping logic, and vice versa. Each layer can evolve independently as long as it adheres to its defined interfaces, which enhances the overall maintainability and extensibility of the system.

In order to allow easy extension of the source parsing layer, the design takes into consideration the Open-Closed principle [66]. According to this principle, software entities (e.g., classes, modules, functions, etc.) should be open for extension but closed for modification. In other words, it means that it should be possible to extend the functionality of a system without making changes to the existing code. In the context of the mapping process, a typical example of this principle is the ability to add support for a new file format (e.g., JSON) by creating a new implementation of the source ingester interface without modifying the existing code that handles CSV files and the ingestion orchestration logic. This design allows the system to adapt to new forensic tools and data formats without risking regressions in existing functionality, which is crucial for a system that needs to evolve over time.

For the object-oriented design choices, the core abstractions are:

- **SourceIngester interface** for format specific readers (e.g., CSV or JSON readers).
- **Mapper interface** for source to knowledge graph transformations.
- **YamlMapper implementation** is a realization of the Mapper interface that interprets YAML mapping rules.
- **SourceRecord interface** that abstracts the structure of a source record as a map of field names to values, decoupling the parsing logic from the mapping logic.
- **IngestionPipeline** that orchestrates the overall process.
- **OntologyFacade and KnowledgeGraphFacade** that provide ontology reading and knowledge graph manipulation.

This design naturally applies common patterns:

- **Strategy pattern:** each concrete SourceIngester implementation encapsulates a specific parsing strategy for a file format, allowing the ingestion pipeline to select the appropriate strategy at runtime based on the source type [67].
- **Registry pattern:** the mappers are registered and stored in a registry, and the ontology, classes, properties are also registered before use.

- **Pipeline pattern:** the ingestion process can be seen as a pipeline where data flows from source files through parsing, mapping, and finally to the knowledge graph serialization, with each stage of the pipeline responsible for a specific transformation. The ingestion pipeline is executing stages in a sequence.
- **Interpreter pattern:** YamlSourceMapper can be seen as an interpreter that reads the YAML configuration and executes the mapping rules to map source records to the knowledge graph in a recursive manner [67].
- **Builder pattern:** the mapping process involves constructing complex objects (individuals and properties in the knowledge graph) based on the source records and the mapping configuration. The mapping logic can be designed to use a builder pattern to incrementally construct these objects while applying the mapping rules. [67].
- **Facade pattern:** the OntologyFacade and KnowledgeGraphFacade are providing a simplified interface to interact with the underlying ontology model and knowledge graph, hiding the complexities of their manipulation and allowing to hide the details of the underlying manipulations.

The architecture is presented in the following UML diagrams and the sequence diagram in Figure 6.2 that illustrates the interactions between the main components during the mapping process.

Figure 6.4 presents the high level architecture of the process. It shows the main function, the starting point and the two other components it uses. The `Loader` loads the ontology file and provides means to initialize the knowledge graph. The `IngestionPipeline` orchestrates the ingestion process and uses two facades for the ontology and knowledge graphs.

About the pipeline, the Figure 6.5 presents its composition. The main abstraction is the `IngestionStage` which is an abstract class representing a stage of the ingestion. Classes extending it need to override the `execute()` method. The `PipelineContext` is available to every stage and contains shared information between stages. An `IngestionTask` is created for each file to ingest and its mapper.

The pattern depicted in Figure 6.6 allows the ingestion of files in multiple formats in an abstract way. Indeed, each file format is an implementation of the interfaces. The other classes using the interfaces do not need to account for file formats.

The mappers configuration is not restricted to a format. The configuration uses generic classes and interfaces, allowing it to be extended to another file format (e.g., JSON, etc.). This is shown in Figure 6.7.

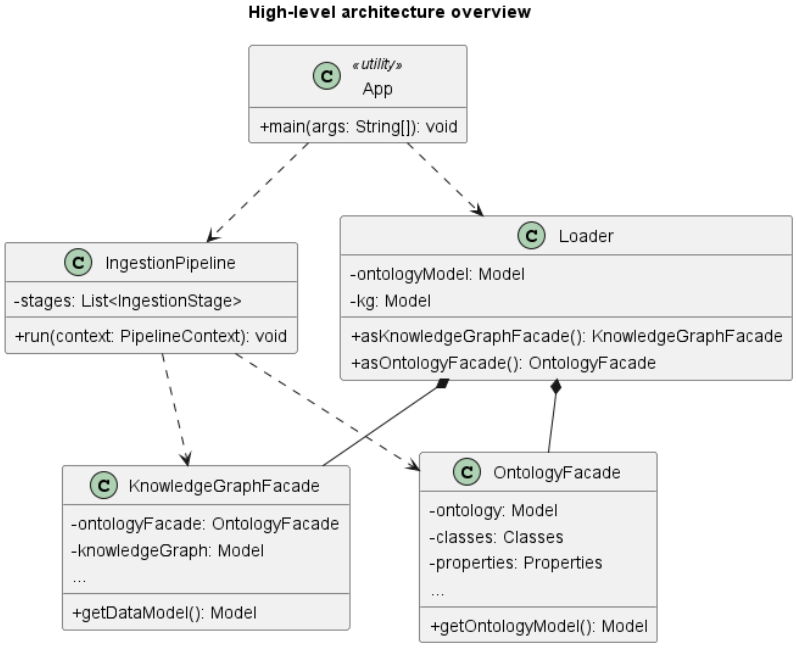


Figure 6.4.: High-level architecture overview of the mapping process

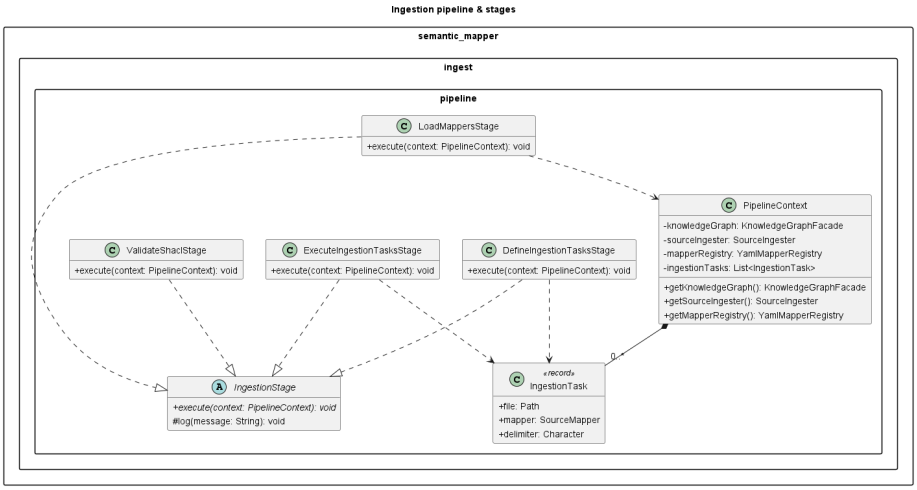


Figure 6.5.: Ingestion pipeline and stages

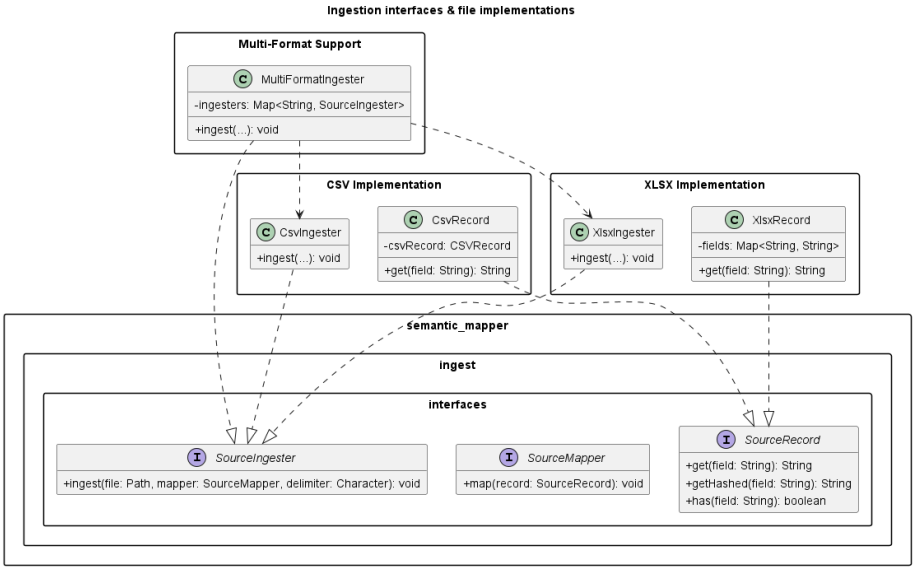


Figure 6.6.: Ingestion interfaces and implementations

This work uses YAML for the mappers configuration file format. To do so, it uses the classes in Figure 6.7 to specialize them for YAML. This is shown in Figure 6.8.

Lastly, Figure 6.9 shows the wrappers around the ontology and knowledge graph. They provide methods to query the ontology and manipulate the knowledge graph.

The complete class diagram is available in the `semantic-mapper` directory of the project repository [57, 58] or through this link

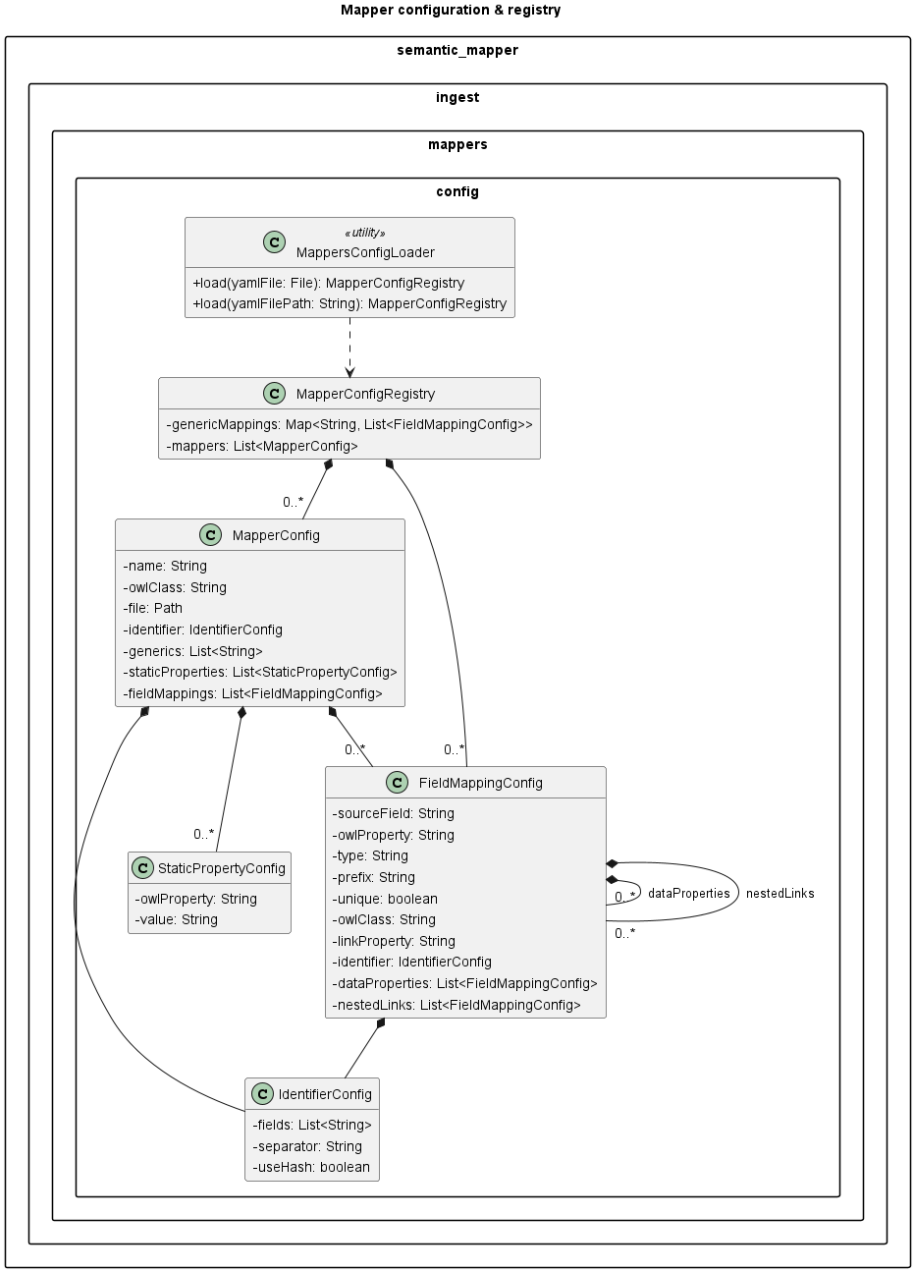


Figure 6.7.: Mapper configuration and registry

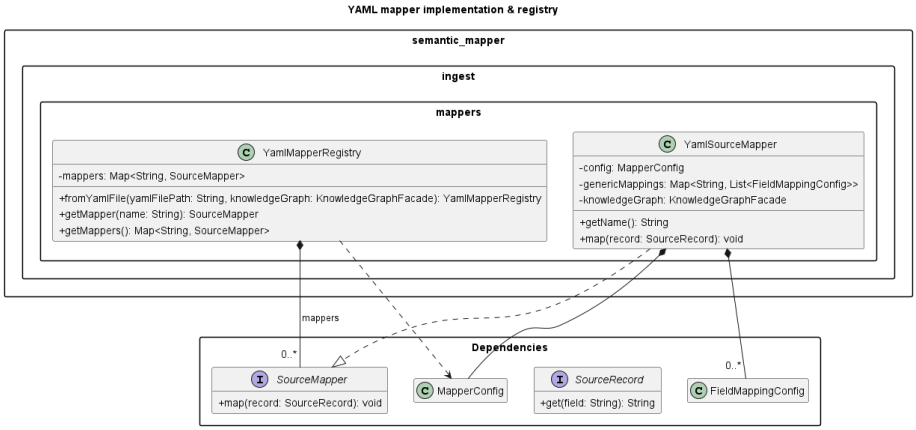


Figure 6.8.: YAML mapper implementation and registry

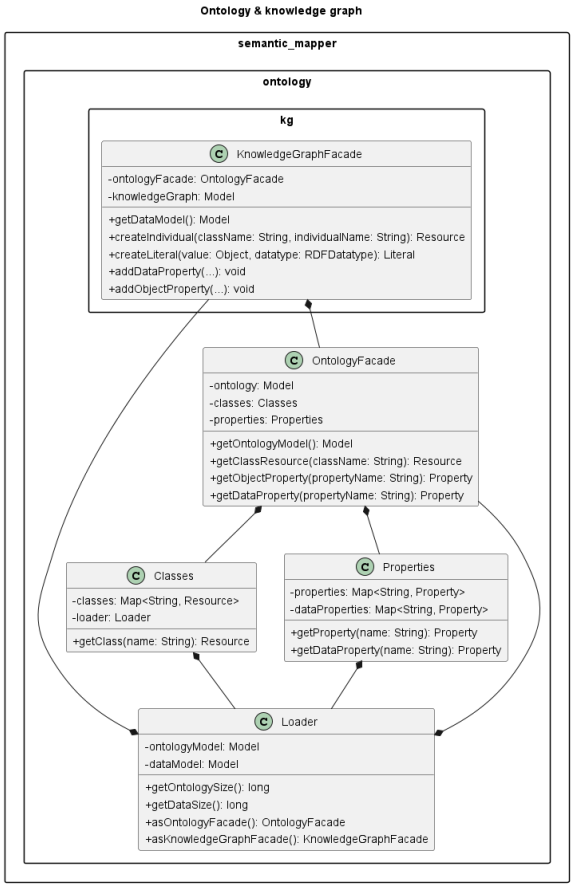


Figure 6.9.: Ontology and knowledge graph wrappers

6.3. Summary

The mapping layer acts as a semantic bridge between raw forensic source files and a knowledge graph. It translates heterogeneous data into ontology compliant RDF triples, making the data meaningful and queryable.

Java was chosen for its mature RDF/OWL libraries (Apache Jena), strong typing, broad format support, and platform independence. This suits the cloud pipeline deployment context and the scale of forensic datasets.

An initial approach of writing one Java class per source file was abandoned as unmaintainable. The final design is configuration-driven: a generic Java engine reads YAML files that declare how each source should be parsed and mapped, making it easy to add new sources or adapt to format changes without touching the core code.

The custom design was developed before the RML/YARRRML frameworks were identified, yet it independently converged on similar principles: declarative mappings, separation of parsing from semantic transformation, and schema-driven graph generation. One notable advantage over YARRRML is native support for generic/reusable mappings. In future iterations, adopting RML/YARRRML could improve interoperability with the broader Semantic Web ecosystem.

The mapping process follows a pipeline pattern. The stages are task definition, loading, mapping, and validation. First, each source record is normalized into a shared structure. Then, a YAML-driven mapper assigns the record a stable identifier, instantiates the appropriate ontology class, populates properties, and links related individuals. Shared mapping fragments prevent duplication across similar source files. Finally, the graph is validated against SHACL constraints before being serialized to RDF/Turtle.

The implementation follows the Single Responsibility principle and the Open-Closed principle. This provides clear interfaces for ingesters, mappers, and records. Applied common design patterns include Strategy, Registry, Pipeline, Interpreter, Builder, and Facade.

Chapter 7 Visualization

The visualization component presents the results of the unified knowledge graph to the user in an intuitive and actionable manner. This includes generating timelines of events and providing interactive exploration and analysis tools. On Figure 7.1, it corresponds to the ARA Knowledge Consumption Layer. This chapter directly addresses RQ2.

The source code of the visualization layer is publicly available in the `UniForeXT-viewer` directory of the project repository [57] and has been archived on Zenodo for long-term preservation [58].

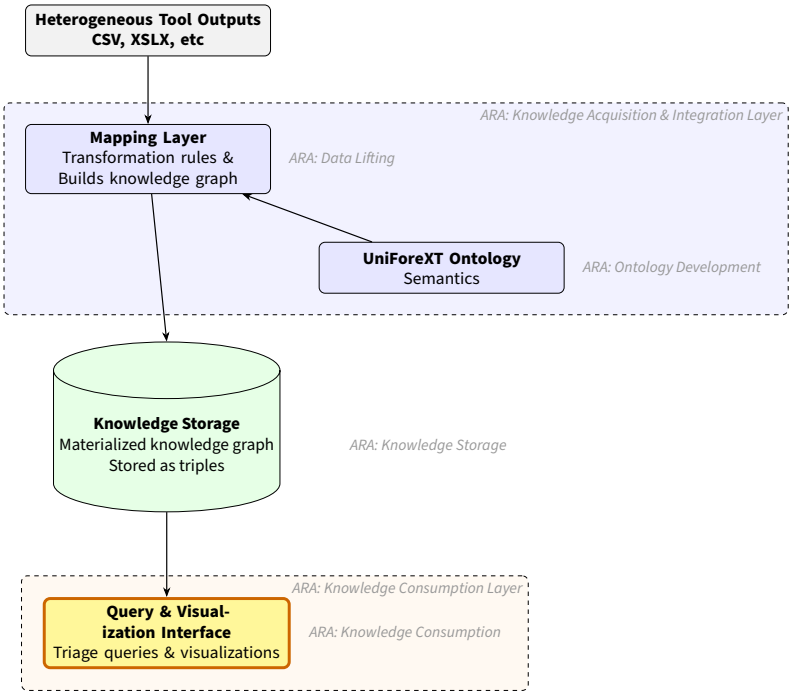


Figure 7.1.: System architecture with the visualization interface highlighted

7.1. Programming Language

The selection of a programming language for the visualization layer is driven by multiple factors: the need for interactive and dynamic visualizations, the availability of libraries for

knowledge graph manipulation and visualization, and support for multiple operating systems and architectures.

Considering all these criteria, Java was selected as a language with extensive support for working with graphs using Jena [63] and visualization using JavaFX [68]. Moreover, Java's cross-platform nature guarantees portability across different platforms without any problems.

7.2. Design

The visualization component is designed as a modular desktop application that separates navigation, data processing, and presentation. The Model-View-Presenter (MVP) pattern is used to structure the application. The model represents the underlying data, which is the knowledge graph. The view is responsible for rendering the user interface and visualizations. The presenter acts as an intermediary that handles user requests, interacts with the model to retrieve the correct data, and provides the view with the necessary information to update the displayed content. This separation of concerns enables a maintainable and extensible codebase where changes to the presentation layer do not affect the data processing logic and vice versa.

A hybrid UI approach combines the responsiveness of a desktop application with the flexibility and availability of graph libraries of web technologies. JavaFX provides the outer application shell and window management, while the internal visualizations are rendered as HTML/JavaScript content inside WebView components. The modular design allows for future extensions, such as adding new views or integrating additional data sources, without significant changes to the overall architecture.

The knowledge graph is loaded once from an RDF file. Either the user provides the location of the file as a command-line argument or through the OS interface. The service layer provides higher-level methods used by the views.

To keep navigation responsive, frequently used results are cached. A ColorService is used to assign stable colors to ontology types, which improves readability and makes repeated categories visually consistent across views.

Being only a frontend application, the visualization components are not responsible for data processing. The design is simpler and does not require advanced design patterns. The main focus is on providing an intuitive and efficient user interface for exploring the knowledge graph, while the underlying data processing and analysis are handled by separate components in the architecture.

7.3. Implementation

The visualization component is structured around the main application class, `App`, which initializes the JavaFX application and manages the different views. The `KGService` class is responsible for loading the knowledge graph from a Turtle file and providing methods for querying and manipulating the graph data.

Since the app makes use of WebView components to render HTML/JavaScript visualizations, the implementation includes a set of HTML templates that are loaded by `HtmlLoader`. The HTML templates are designed to render the visualizations using advanced JavaScript libraries, such as Chart.js [69] for the different charts and vis.js [70] for the interactive graph visualizations.

The core logic is concentrated in `KGService`. This class loads the RDF and provides methods to detect relevant date properties, extract timeline items, identify undated individuals, compute graph neighborhoods and summaries, etc. It leverages the Apache Jena [63] library for RDF processing. It also provides a small set of utility methods, such as local-name extraction. The service caches some computed results to avoid repeating expensive model traversals, particularly expensive SPARQL aggregation queries. The service is designed to be reusable across different views, providing a consistent interface for accessing the knowledge graph data. The implementation focuses on efficiency and responsiveness, ensuring that the visualization components can quickly retrieve and display the necessary information without significant delays.

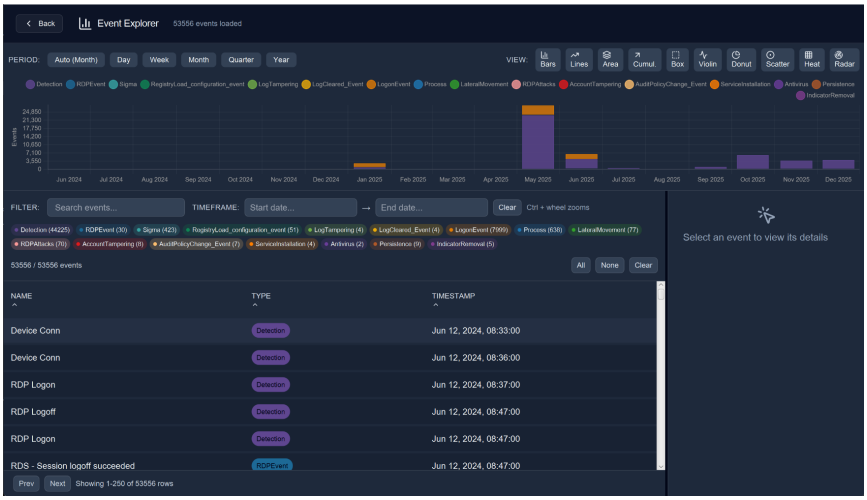
The communication between the Java and JavaScript is possible with JavaBridge. This bridge exposes methods to JavaScript running inside the WebView and acts as the communication layer between the HTML pages and the Java code. Each view loads its corresponding HTML resource through `HtmlLoader`, injects the bridge into the page context, and initializes the interactive visualization once the page has finished loading.

7.4. Views

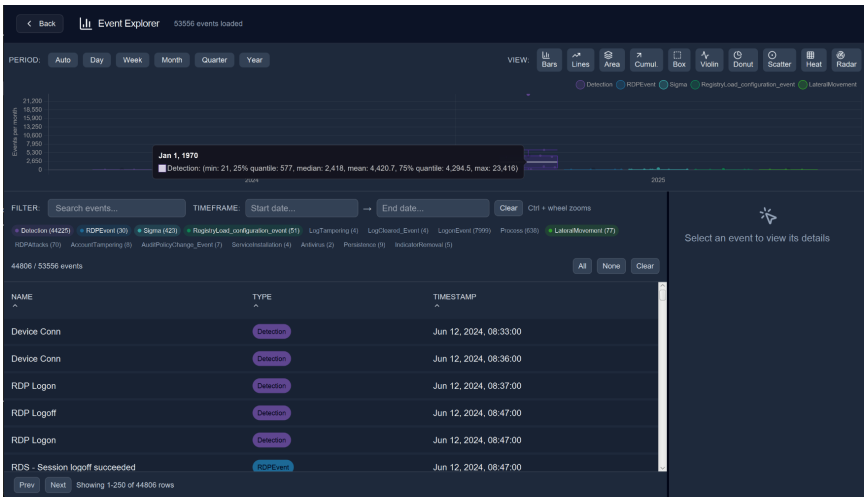
The visualization component provides multiple complementary views designed to support different exploration patterns and analytical tasks. Each view offers a distinct perspective on the knowledge graph, from high-level temporal patterns to detailed contextual explorations. Together, they enable users to discover insights, investigate specific events, and perform custom queries through an intuitive interface.

7.4.1. Event explorer view

This view provides a histogram of events over time, allowing users to quickly identify periods of high activity on an endpoint. The histogram is interactive, enabling users to define specific time ranges. Other analysis views are available to explore the events with different analytical displays, such as cumulative counts, box plots, violin plots and heatmaps. These charts are built with Chart.js [69]. Figure 7.2 shows the view with the histogram and box plots charts.



(a) with histogram



(b) with box plots

Figure 7.2.: Screenshots of the event explorer view

Under the different charts, a list of events is displayed, showing their information, such as timestamp, name, and type. The list is also interactive, allowing users to click on an event to highlight its position in the chart and display information in the right column. When elements of the charts are clicked, the list of events goes to the first event of the corresponding clicked

chart. Users can double-click on an event to open a detailed graph view centered around the selected event, showing its relationships and context within the knowledge graph. This view is defined in Subsection 7.4.3.

In this view, users can also apply filters (e.g., inclusion, exclusion, range) to focus on specific types of events, time ranges, or other criteria. The filters are applied across all charts and the event list, allowing for a cohesive exploration experience. This view is particularly useful for quickly identifying patterns and anomalies in the event data, as well as for drilling down into specific time periods of interest.

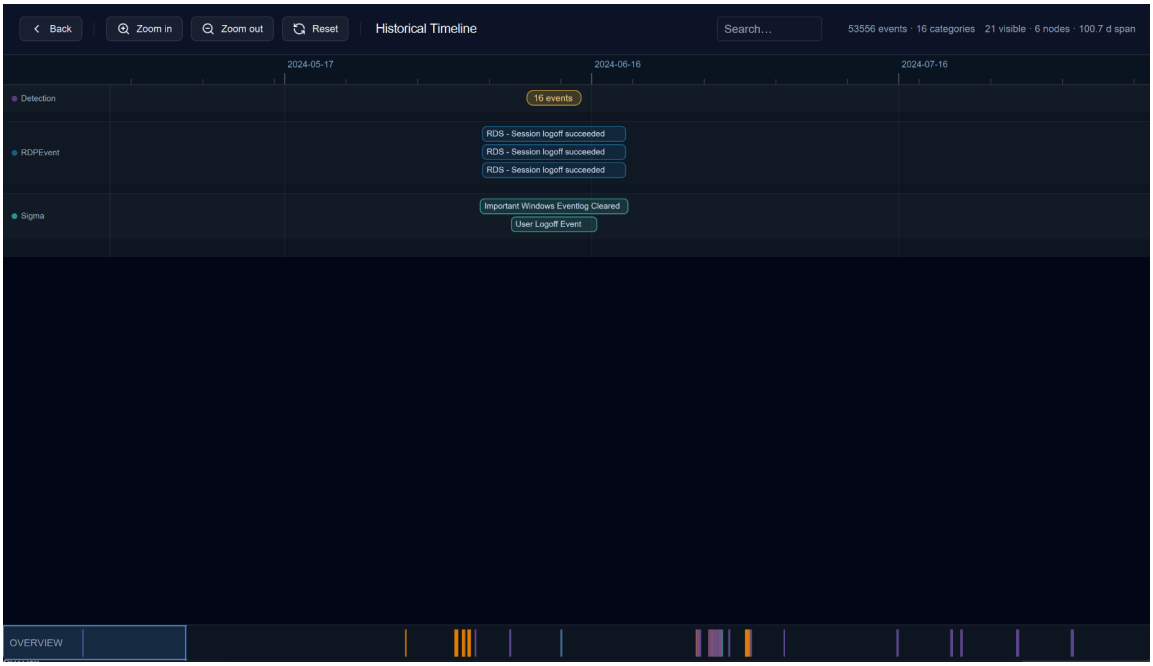
7.4.2. Timeline view

The timeline view provides a chronological representation of events. It allows analysts to inspect the sequence of events and their temporal relationships. Users can zoom in and out of the timeline to focus on specific time periods or to get an overview of the entire timeline. This view is particularly useful for understanding the order of events and identifying any temporal patterns or correlations between different types of events. The timeline can also be filtered based on event types, time ranges, or other criteria, allowing users to focus on specific subsets of the data. Given the high quantity of data, the timeline view spans vertically, allowing users to scroll through the events while keeping the temporal context intact. Screenshots of the timeline view are shown in Figure 7.3.

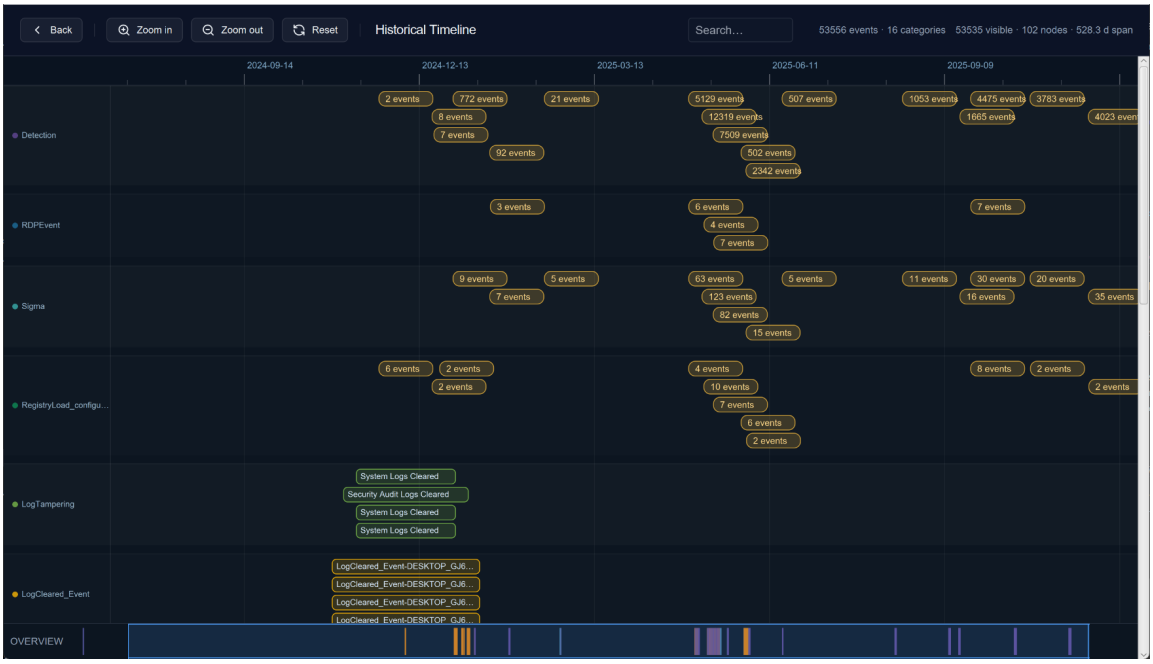
Below the timeline, a temporal navigation panel is provided, showing a condensed overview of the entire timeline. This panel allows users to quickly move their focus to different time periods. The time window currently selected is indicated in the navigation panel and can be moved around to change the time period. The entire list of events within the timeline is shown in the navigation panel through small vertical lines.

Like for the event explorer, users can double-click on an event in the timeline to open a detailed graph view centered around the selected event, showing its relationships and context within the knowledge graph. This view is defined in Subsection 7.4.3.

The timeline view was implemented with the assistance of LLMs because of the performance requirements that emerged early in the development of this view. An initial attempt using vis.js revealed significant rendering and interaction bottlenecks caused by the high node count. This issue highlights how library abstractions can limit performance in specific scenarios due to the generic nature. The timeline was ultimately built from scratch, enabling fine-grained, tailored performance optimizations.



(a) with a narrow zoom



(b) with a wide zoom

Figure 7.3.: Screenshots of the timeline view

7.4.3. Graph view

This view provides an interactive graph visualization centered around a selected entity. It displays the relationships and context of the entity within the knowledge graph, allowing users to explore connected entities. Users can double-click on the nodes to expand the network around the selected node. The graph visualization is meant to enable users to comprehend the context of particular events and their connection to other entities in the knowledge graph. Two examples of graph views are shown in Figure 7.4.

In case of neighbors with a large number of connections, the graph view will display a summary of the neighborhood and allow the user to select a subset of the neighbors to display. This helps to manage the size of the graph and maintain readability while still providing access to relevant information.

7.4.4. Query view

The query view allows users to execute SPARQL queries against the knowledge graph and visualize the results. These queries are limited to read-only queries. The view provides an interface for users to input their SPARQL queries and displays the results in a tabular format. Users can double-click on any result to open the graph view centered around the selected result.

Multiple prefixes are appended to queries for user convenience. These are shown in a box below the query form.

Figure 7.5 shows an example of a query using the query view.

7.4.5. Undated items view

The undated items view lists all items that are not represented in the timeline and event explorer views. This view displays items based on their categories. Figure 7.6 shows its layout.

(a) from a computer entity

(b) from a detection

Figure 7.4.: Screenshots of the graph view

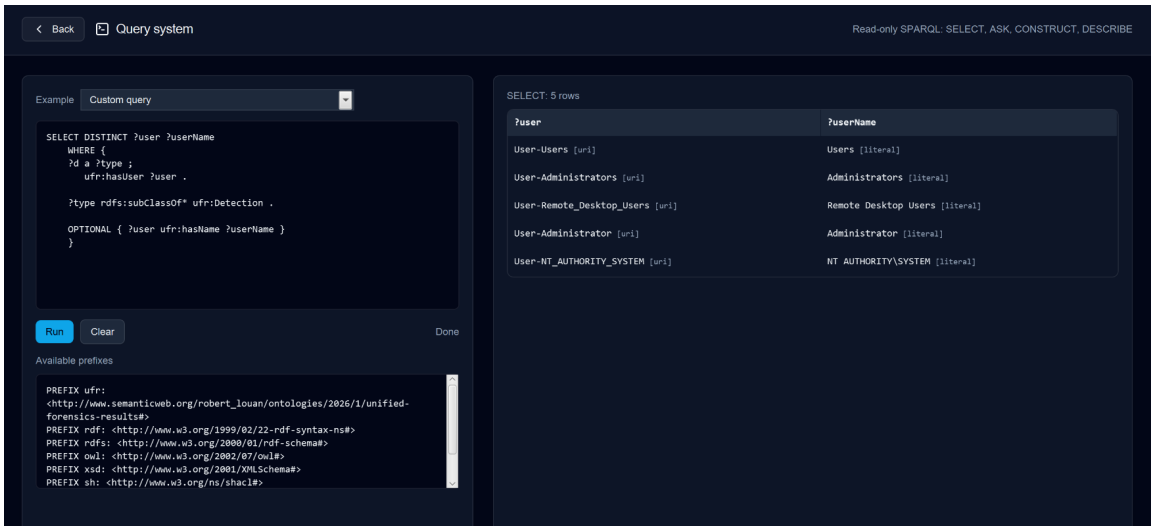


Figure 7.5.: Screenshot of the query view

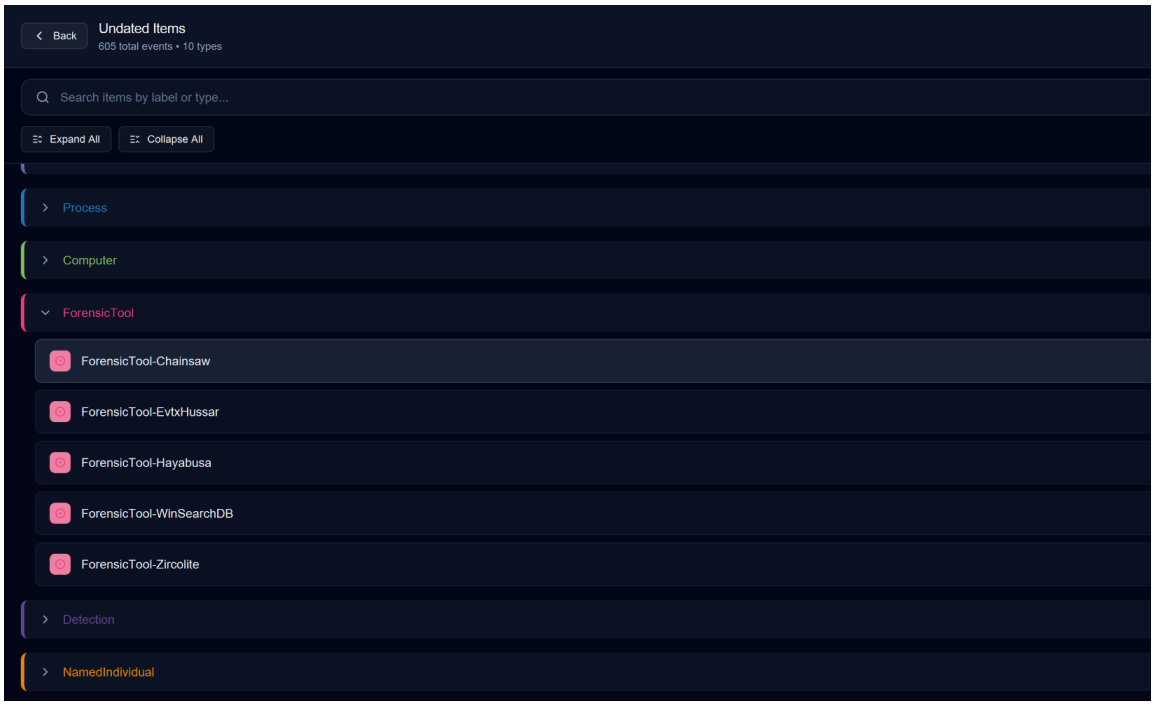


Figure 7.6.: Screenshot of the undated items view

7.5. Requirements coverage

The visualization component directly addresses RQ2 and is designed to meet the functional requirements defined for the visualization interface. Specifically, it supports cross-tool navigation (FR7) by allowing users to explore entities and events across tool boundaries from a single view. The interactive filtering and scoping capabilities (FR8) enable users to focus on specific time ranges, endpoints, or other criteria relevant to their analysis. Finally, the various visualization capabilities (FR9), including timelines, charts, and graph visualizations, provide analysts with multiple ways to explore and understand the data within the unified model.

7.6. Summary

This chapter presented the visualization component of the system. It is implemented as a modular desktop application in Java using JavaFX with HTML/JavaScript rendering. The MVP architectural pattern was adopted to cleanly separate data access, business logic, and presentation concerns. Using a hybrid UI approach enabled the integration of rich browser-based visualization libraries within a distributable application.

The `KGService` class serves as the central data access layer, loading the RDF graph from a Turtle file and exposing cached, higher-level query methods consumed by all views. Communication between the Java backend and the JavaScript frontends is handled through the `JavaBridge`, which exposes a callback interface for user interactions such as node selection and double-click navigation.

Four complementary views were introduced. The Event Explorer provides a histogram temporal analysis with interactive filtering and multiple chart types. The Timeline View offers a chronological, scrollable representation of events with a condensed navigation panel. Given performance constraints at scale, it was built from scratch rather than relying on existing libraries. The Graph View enables contextual exploration of individual events and their neighborhood within the knowledge graph, with summarization support for highly connected nodes. Finally, the Query View exposes direct SPARQL querying with tabular results and seamless integration into the graph view.

Together, these views offer an intuitive and efficient environment for exploring the knowledge graph. They support high-level pattern discovery, detailed event-level investigations, and custom queries.

Chapter 8 Validation

During the ontology validation (see Section 5.5), the ontology was evaluated against competency questions (CQs) using authoring tests derived from the methodology proposed in [47]. To go further and show that the ontology can actually answer the CQs, the SPARQL query language will be used to query the ontology and check if the expected answers are returned. Next, we will validate the different functional and non-functional requirements defined in Chapter 3. Finally, we will check the coverage of the requirements by the ontology.

8.1. System Validation

The system validation will be performed by executing SPARQL queries corresponding to the competency questions defined in Chapter 3. The results of these queries will be analyzed to determine if the ontology can effectively answer the questions. This section will present the queries and the dataset used for testing.

8.1.1. Competency questions to queries

For almost every CQ, its corresponding SPARQL query is defined. For some CQs, a SPARQL query is not needed because the question can be answered visually with the visualization views defined in Chapter 7.

? SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is the standard query language and protocol for retrieving and manipulating data stored in the Resource Description Framework (RDF) format [71]. It is standardized by the World Wide Web Consortium (W3C) and has become the primary means of querying knowledge graphs.

RDF represents information as triples of the form `(subject, predicate, object)`, forming a directed labeled graph. SPARQL exploits this structure by allowing queries to define graph patterns against such triples. The fundamental building block of a SPARQL query is the *Basic Graph Pattern* (BGP), which is a set of triple patterns in which some positions may be replaced by variables (denoted with a `?` prefix). The query

engine then binds those variables to values from the dataset such that the resulting triples exist in the graph.

A SPARQL SELECT query follows a structure analogous to SQL: the SELECT clause lists the variables to project, the WHERE clause defines the graph pattern to match, and optional modifiers such as FILTER, OPTIONAL, ORDER BY, and LIMIT refine the result set. For example, the following query retrieves all entities classified as a given type along with their labels:

```
SELECT ?entity ?label
WHERE {
    ?entity rdf:type  ex:SomeClass ;
            rdfs:label ?label .
    FILTER (lang(?label) = "en")
}
ORDER BY ?label
```

Beyond SELECT, SPARQL 1.1 defines three additional query forms: CONSTRUCT, which returns a new RDF graph synthesized from query results; ASK, which returns a boolean indicating whether a pattern has at least one solution; and DESCRIBE, which returns an RDF description of a resource. SPARQL 1.1 also introduced update operations (INSERT, DELETE, LOAD) for modifying RDF datasets, as well as federated queries via the SERVICE keyword, enabling a single query to span multiple remote SPARQL endpoints.

In the following SPARQL queries, we assume that the ufr: prefix is defined as the UniForeXT ontology and that other standard prefixes are defined (rdf:, rdfs:, etc.)

1. Are there any detection events on endpoint X?

```
ASK WHERE {
?d a ufr:Detection ;
   ufr:hasComputer ?c .
?c ufr:hasHostname "HOSTNAME" .
}
```

2. How many info, low, medium, high detection events on endpoint X?

```
SELECT ?level (COUNT(?d) AS ?count)
WHERE {
  ?d ufr:hasLevel ?level .
  ?c ufr:hasHostname "HOSTNAME" .
```

```
}
GROUP BY ?level
ORDER BY DESC(?count)
```

3. Which users are associated with suspicious events on endpoint X?

```
SELECT DISTINCT ?user ?userName
WHERE {
?d a ?type ;
   ufr:hasUser ?user .

?type rdfs:subClassOf* ufr:Detection .
?c ufr:hasHostname "HOSTNAME" .
OPTIONAL { ?user ufr:hasName ?userName }
}
```

4. Which suspicious processes were executed on endpoint X?

```
SELECT DISTINCT ?proc ?procName ?procPath
WHERE {
?d a ?type ;
   ufr:hasProcess ?proc .
?c ufr:hasHostname "HOSTNAME" .

?type rdfs:subClassOf* ufr:Detection .

OPTIONAL { ?proc ufr:hasName ?procName }
OPTIONAL { ?proc ufr:hasPath ?procPath }
}
```

5. When did the first detection occur on endpoint X?

```
SELECT ?d ?ts
 WHERE {
   ?d a ufr:Detection ;
      ufr:hasComputer ?c ;
      ufr:hasTimestamp ?ts .
   ?c ufr:hasHostname "HOSTNAME" .
 }
 ORDER BY ASC(?ts)
 LIMIT 1
```

6. Where did detection X originate?

```
SELECT DISTINCT ?neighbor
WHERE {
  {
    <DETECTION-IRI> ?p1 ?mid .
    ?mid ?p2 ?neighbor .
  }
  UNION
  {
    ?neighbor ?p1 ?mid .
    ?mid ?p2 <DETECTION-IRI> .
  }
}
```

7. Which detections are associated with attack tactic/technique X?

```
SELECT ?d ?ts
 WHERE {
   ?d a ?type ;
      ufr:hasAttackTactic <TACTIC_URI> ;
      OPTIONAL { ?d ufr:hasTimestamp ?ts }

   ?type rdfs:subClassOf* ufr:Detection .
 }
```

8. How many attack techniques and tactics are present?

```
SELECT ?type (COUNT(DISTINCT ?instance) AS ?count)
WHERE {
  {
    ?instance a ufr:AttackTechnique .
    BIND("AttackTechnique" AS ?type)
  }
  UNION
  {
    ?instance a ufr:AttackTactic .
    BIND("AttackTactic" AS ?type)
  }
}
GROUP BY ?type
```

9. Which endpoints show evidence of X technique?

```
SELECT ?entity ?technique ?techniqueLabel ?techniqueID
WHERE {
    ?entity ufr:hasAttackTechnique ?technique .
    ?technique a ufr:AttackTechnique .
    OPTIONAL { ?technique rdfs:label ?techniqueLabel . }
    OPTIONAL { ?entity ufr:hasTechnique ?techniqueID . }

    FILTER (
        CONTAINS(LCASE(STR(?techniqueLabel)), LCASE("INSERT LABEL"))
        || CONTAINS(LCASE(STR(?techniqueID)), LCASE("INSERT ID"))
    )
}
```

10. Which endpoints show evidence of X tactic?

```
SELECT ?entity ?tactic ?tacticLabel
WHERE {
    ?entity ufr:hasAttackTactic ?tactic .
    ?tactic a ufr:AttackTactic .
    OPTIONAL { ?tactic rdfs:label ?tacticLabel . }

    FILTER (CONTAINS(LCASE(STR(?tacticLabel)), LCASE("INSERT LABEL")))
}
```

11. From which source tool and source file was each normalized entity derived?

```
SELECT ?target
WHERE {
  <ARTIFACT-IRI> ufr:hasForensicTool ?target .
}
```

12. Which original raw records justify a given normalized detection ?

```
SELECT ?target ?path ?pathlit
WHERE {
    <ARTIFACT-IRI> ufr:hasLogFile ?target.
    ?target ufr:hasPath ?path.
    ?path ufr:hasContent ?pathlit.
}
```

8.1.2. Dataset

The evaluation of the SPARQL queries, visualization views, and overall ontology functionality relies on three synthetic datasets provided by CERT Belgium. These datasets capture realistic endpoint activity, encompassing diverse processes, user interactions, system events, and log entries. While not specifically designed to assess this work, they constitute a representative basis for testing the proposed framework.

Each dataset contains outputs from 6 different forensic tools, including Hayabusa [7], Chainsaw [9], etc. Each tool generates various artifacts, like detections and events, using different schemas and file formats, such as CSV and XLSX. The average size per dataset is around 500 megabytes.

8.1.3. Query results

The visualization results can be found in Chapter 7. The SPARQL query results are described below, and an example of the results for each query is provided.

1. **Are there any detection events on endpoint X?**
   Returned `true` when there are detections for the given endpoint.
2. **How many {info, low, medium, high} detection events on endpoint X?**
   Returned a table with the count of detections for each level.

?level	?count
info	5
low	10
medium	15
high	8

3. **Which users are associated with suspicious events on endpoint X?**
   Returned a list of users with their names (if available) who are associated with detections on the given endpoint.

?user	?userName
User-Users	Users
User-Administrators	Administrators
User-Remote_Desktop_Users	Remote Desktop Users
User-Administrator	Administrator
User-NT_AUTHORITY_SYSTEM	NT AUTHORITY\SYSTEM

4. **Which suspicious processes were executed on endpoint X?**
   In the current datasets, no process information is available for the detections, so this query returned an empty result set. Nonetheless, the query is correctly formulated and stays valid for future datasets that may include process information.

5. **When did the first detection occur on endpoint X?**
   It returns the earliest timestamp associated with a detection on the given endpoint, along with the corresponding detection identifier.

?detection	?ts
Detection-1_MS_Win_Partition_Diagnostic	2024-06-12T08:33:00Z

6. **Where did detection X originate?**
   It returns a list of neighboring entities (e.g., related processes, users, files) that are connected to the specified detection in the knowledge graph. Double-clicking on a node in any view results in executing this query and showing the neighboring entities, which can be of various types such as forensic tools, computers, classes, etc.

?neighbor
66a8b0c68266578d3afef21c7413dfea
a6ba8e28d1c53dc0f41b1794cc1272b8
ForensicTool
WIN-MT7VOSGDTL1
Computer
Class
Independent_entity
...

7. **Which detections are associated with attack tactic/technique X?**
   The query returns a list of detections that are linked to the specified attack tactic or technique, along with their timestamps if available. The dataset used does not allow testing this query, as it does not contain easily extractable information about attack techniques and tactics.

8. **How many attack techniques and tactics are present?**
   Same as above, the dataset used does not allow us to test this query. It would return a count of distinct attack techniques and tactics present in the knowledge graph.

9. **Which endpoints show evidence of X technique?**

10. **Which endpoints show evidence of X tactic?**
    For both queries (9 and 10), the dataset used does not allow testing. They return a list of entities (e.g., computers, users) that are associated with the specified attack technique or tactic, along with the technique/tactic labels and identifiers if available.

11. **From which source tool and source file was each normalized entity derived?**
    It returns the forensic tool and log file associated with a given normalized entity (e.g., detection, event) in the knowledge graph. This allows tracing back the provenance of the normalized data to its original source.

?target
ForensicTool-Hayabusa

12. **Which original raw records justify a given normalized detection?**
    It returns the original log file that was used by the forensic tool to derive the artifact.

?target	?path	?pathlit
LogFile-...67b2	Path-...67b2	F:\c\windows\system32\...\Archive-Security...evtx

8.1.4. Execution time and performance

To assess whether the system meets NFR4, performance was evaluated at two layers: the data transformation and mapping pipeline, and the execution of SPARQL queries against the populated knowledge graph. All measurements were taken on a machine equipped with an Intel Core i7-13850HX (20 cores, 28 threads, 2.10 GHz base clock) and 32 GB of DDR5 memory (5600 MT/s). Each measurement represents the mean of three consecutive runs to reduce the effect of transient system load.

The mapping pipeline, excluding SHACL validation, completes in a mean of 31.47 seconds across the three datasets. Including the SHACL validation it completes in 2 minutes and 12 seconds. Given that this process ingests approximately 500 MB of heterogeneous forensic data from six tools, produces a fully populated knowledge graph and can be embedded inside an existing pipeline, this duration is acceptable for a triage context where the mapping is performed once per investigation, not interactively.

SPARQL query execution times are reported in Table 8.1. All queries return results in under 300 ms, confirming that interactive exploration of the knowledge graph is feasible within operational time constraints.

#	Query	Mean execution time
1	Detections on endpoint X?	4 ms (true)
1	Detections on endpoint X?	8 ms (false)
2	Count by severity level	36 ms
3	Associated users	67 ms
5	First detection timestamp	266 ms
6	Detection neighborhood	4 ms
11	Provenance: source tool	2 ms
12	Provenance: raw log record	1 ms

Table 8.1.: SPARQL queries and their mean execution time

8.1.5. Discussion

The results demonstrate that the ontology successfully answers the majority of the competency questions defined in Chapter 5. Queries 1 to 3, 5, and 6 returned meaningful results, confirming that the ontology correctly models the data. The provenance queries (11 and 12) are particularly noteworthy, as they validate the full traceability chain from a normalized detection back to its raw log file and originating forensic tool.

However, several queries could not be fully evaluated due to limitations of the dataset. Queries 7 to 10 returned no results. These queries concern ATT&CK tactics and techniques. This is because the forensic tool that provides attack techniques and tactics is providing them inside multi-line strings. This is a dataset limitation rather than an ontological one. The relevant classes and properties (`ufr:AttackTechnique` and `ufr:AttackTactic`) are defined in the ontology. Similarly, query 4 returned an empty result set due to the absence of process and detection relations.

8.2. Requirements Coverage

The functional and non-functional requirements are covered as follows:

- **Functional Requirements:**
  - **FR1 - Multi-source ingestion:** this requirement is validated by the successful integration of data from multiple forensic tools into the dataset, demonstrating that the ontology can accommodate heterogeneous inputs.
  - **FR2 - Unified representation:** this is validated by the ability to query across different tools and data types using a unified set of classes and properties.

  - **FR3 - Integration support:** this requirement is satisfied by the ontology's native support for cross-tool entity integration via shared IRI references. Covering differently named entities that refer to the same concept is not supported. Several implementation paths exist for future work (e.g., rule-based inference, embedding-based similarity). Integration of identically named entities carrying divergent or updated information is natively supported.
  - **FR4 - Queryability:** this is validated by the successful execution of SPARQL queries and the different views defined in Chapter 7 that allow users to explore the data and answer the competency questions.
  - **FR5 - Visualization readiness:** this requirement is validated by the different visualization views defined in Chapter 7 that allow users to explore the data and answer the competency questions.
  - **FR6 - Traceability:** this is validated by the successful execution of provenance queries (11 and 12) that demonstrate the ability to trace normalized entities back to their source tools and raw log files.

- **Non-Functional Requirements:**
  - **NFR1 - Tool-agnostic design:** this requirement is validated by the ontology's ability to represent data from various forensic tools without relying on tool-specific schemas, as demonstrated by the successful integration of multiple tools into the dataset.
  - **NFR2 - Extensibility:** this requirement is validated by the schema's ability to allow the addition of new artifact and entity types with minimal impact on existing mappings.
  - **NFR3 - Interoperability:** this requirement is validated by the model's reliance on explicit, documented semantics to reduce ambiguity across tools.
  - **NFR4 - Performance for triage:** this requirement is validated by Subsection 8.1.4: the mapping pipeline processes the data in 31.47 seconds, and all queries return results in under 300 ms, confirming operational viability.
  - **NFR5 - Data quality robustness:** this requirement is validated by the model's ability to tolerate incomplete, noisy, and partially inconsistent forensic data.
  - **NFR6 - Model quality:** this requirement is validated by the model following best practices, including proper denormalization and avoidance of common modeling pitfalls (using OOPS! [59]).
  - **NFR7 - Model correctness:** this requirement is validated by the model being representative of the forensic data it must ingest and being competent in answering

typical triage queries. This is proven by the consumption and mapping layers.

- **NFR8 - Data serialization:** this requirement is validated by the mapping layer serializing the populated knowledge graph to a RDF file, which is a static, self-contained format suitable for storage on object storage.

The ontology was designed to meet the functional and non-functional requirements outlined in Chapter 3. The validation process confirms that the ontology effectively addresses these requirements, particularly in terms of data integration, provenance tracking, and support for reasoning. The competency questions that were successfully answered demonstrate the ontology's capability to represent and query complex relationships between forensic artifacts, tools, and attack techniques. The limitations observed in certain queries highlight areas for future dataset enhancement, rather than deficiencies in the ontology itself. Overall, the validation results indicate that the ontology provides a robust framework for modeling and analyzing forensic data in a structured and semantically rich manner. The requirement FR3 is validated by the ability to integrate artifacts sharing the same identifier but covering differently named identities that refere to the same underlying concept is not supported (e.g., two different user names referring to the same underlying user)

8.3. Summary

This chapter validated the UniForeXT ontology by defining and executing SPARQL queries derived from the competency questions and by checking requirement coverage using the datasets. The evaluation shows that core triage queries (detections, severity counts, associated users, earliest detection, and detection neighborhood) and provenance tracing are supported by the ontology. All queries return results in under 300 ms and the mapping process completes in under 32 s. Queries relying on process details or ATT&CK mappings could not be evaluated because the datasets lack extractable process fields and structured ATT&CK annotations. These are dataset limitations rather than ontological issues. Overall, this chapter provides initial evidence that the ontology satisfies the core functional and non-functional requirements. The CQs that were used to build queries and their results are summarized in Table 8.2.

Table 8.2.: Competency questions results

#	Competency Questions	Result
1	Detections on endpoint X?	✓
2	Count by severity level	✓
3	Associated users	✓
4	Suspicious processes	**(!)** Not present in dataset
5	First detection timestamp	✓
6	Detection neighborhood	✓
7–10	ATT&CK tactic/technique queries	**(!)** Not present in dataset
11	Provenance: source tool	✓
12	Provenance: raw log record	✓

Chapter 9 Conclusion and Future Work

9.1. Conclusion

This thesis introduces UniForeXT, a practical approach to unify heterogeneous forensic tool outputs into a queryable knowledge graph in the context of incident triage. UniForeXT combines a lightweight ontology, a configuration-driven YAML mapper that produces stable identifiers, SHACL-based validation, and an interactive visualization layer to help analysts rapidly find, correlate, and explore high-value artifacts across multiple tools.

These components provide answers to the two research questions presented in the introduction (see Chapter 1). For RQ1, an ontology was found to be the most suitable foundation for integrating heterogeneous data from multiple forensic tools. The ontology preserves the provenance of the data and enables triage-oriented queries over heterogeneous artifacts. Knowledge graphs support quick multi-hop relationships and other operations used for triage queries. Regarding RQ2, the visualization component allowed analysts to view the constructed knowledge graph from different perspectives, such as timelines, graphs, and SPARQL query views.

At the core of this work is the UniForeXT ontology. It provides a unified model to represent forensic tool outputs. The ontology was designed with a focus on triage operations, as well as extensibility to support new tools and data sources. This ontology was designed using a combination of competency questions and iterative refinement based on competency questions and sampling of the forensic tools. It was evaluated using OOPS!, which revealed issues that were addressed during development. The ontology was validated using authoring tests. They ensure that the ontology can represent the necessary concepts and relationships to support triage operations.

The YAML-based mapping system is built on the ontology. It transforms raw tool outputs into RDF triples that conform to the ontology. The mapping system is not restricted to the UniForeXT ontology and could be use with another one. Therefore, modifications to the UniForeXT ontology do not require changes to the mapping system. The mapping pipeline uses SHACL shapes to validate the generated graph. This ensures that the graph adheres to the ontology model. Relying on configuration rather than code lowers the difficulty to extending the mapper with new tools.

For analysts, the primary interface is a visualization suite comprising timeline, graph, and SPARQL query views. Together these views support a triage workflow that moves from temporal overview to structural exploration to precise evidence retrieval, without requiring analysts to switch tools or reconstruct context between steps. The design reflects a deliberate choice to allow analysts to explore the graph in a way that suits their needs.

Finally, an empirical evaluation established that the system is viable under realistic conditions. The mapping pipeline ingested each evaluated dataset in a mean of approximately 31.47 seconds, and all representative SPARQL queries returned results in under 300 ms, which allows for interactive responses. While these measurements were obtained on a single machine and should not be generalized without further benchmarking, they provide an initial empirical basis for claims about usability that would otherwise rest on design intent alone.

9.2. Limitations

The evaluation revealed several limitations. The datasets used are realistic but not exhaustive, as ATT&CK and process extraction were constrained by source formats, which limited some attack-focused evaluations. Performance and timing measurements were taken on a single machine and may change with different hardware or data volumes. Some SHACL shapes were relaxed to avoid rejecting imperfect forensic data. The semantic mapper relies on stable source identifiers, meaning datasets with volatile or weak identifiers may require additional reconciliation logic. The visualization is designed for triage but may not meet all analyst needs or preferences, and user studies are needed to validate its effectiveness in real-world workflows. Finally, the mapping component was found to partly duplicates functionnality offered by existing KGC frameworks (notably RML/YARRRML). Nonetheless, it was implemented before discovering these frameworks and resembles them in its core principles.

9.3. Reflections

The development of UniForeXT highlighted the importance of data engineering practices in the cybersecurity domain. Even more broadly, during the literature review, I observed that many tools and processes in cybersecurity involve data engineering. This experience strengthened my interest in data engineering and its applications in cybersecurity. I am impressed by the potential of knowledge graphs and ontologies, this work has shown me how they can be applied to solve data integration challenges in a practical domain. Although

these technologies are relatively recent, in my opinion, they have a lot of potential to solve real-world problems in a wide variety of domains.

9.4. Outcomes and lessons learned

This thesis produced several technical, methodological, and personal outcomes. First, I learned that the main difficulty in forensic triage lies not only in the volume of data, but also in the heterogeneity of the outputs produced by different tools. While each tool may be useful on its own, combining their results into a coherent view requires careful modeling, stable identifiers, and explicit semantics. This made it clear that data integration is a central problem in DFIR workflows.

Second, I discovered that ontology design is an iterative engineering activity rather than a purely theoretical modeling exercise. Competency questions, the OOPS! evaluation, authoring tests, SHACL shapes, and practical mapping results all influenced the final model. This process demonstrated that an ontology should be evaluated for both conceptual quality and its ability to support concrete queries on real data.

Finally, comparing relational, document, and graph models reinforced my understanding of how they work and the trade-offs involved.

9.5. Future work

The following extensions would introduce improvements to UniForeXT:

- **Web-based, multi-user visualization:** migrate the visualization to a web app with role-based access control and server-side indexing to support larger graphs and collaborative workflows.
- **Entity reconciliation:** implement a system for reconciliation to link partial and noisy identifiers across tools.
- **extract multi-string content:** the current mapping component operates solely on the top-level file structure (e.g., CSV rows and columns) and cannot map nested or embedded data structures, such as JSON content contained within a CSV field. An extension could allow embedded data structure parsing.
- **KGC interoperability:** evaluate RML/YARRRML or similar frameworks for mapping portability and community tooling.

- **LLM-assisted workflows:** assess whether LLMs benefit more from structured UniForeXT inputs than from raw logs for summarization and analyst assistance.
- **Expanded benchmarking and user study:** evaluate at a larger scale on additional tool outputs and with incident responders to quantify analyst efficiency gains.

9.6. Final remarks

UniForeXT demonstrates that careful ontology design, together with pragmatic mapping and validation practices, can bridge heterogeneous forensic outputs into a queryable and explorable knowledge graph suitable for triage. The visualization interface allows analysts to interact with the resulting knowledge graph through multiple views. These views support a triage workflow that moves from a temporal overview to structural exploration to precise artifact retrieval. While there are limitations and areas for improvement, this work provides a practical foundation for unifying forensic data and supporting analyst workflows in incident response.

Appendix A.

Tables

Table A.1.: CQ Archetypes from [47]

ID	Pattern	Example	PA	RT	M	DE
1	Which [CE1] [OPE] [CE2]?	Which pizzas contain pork?	2	obj.		
2	How much does [CE] [DP]?	How much does Margherita Pizza weigh?	2	data.		
3	What type of [CE] is [I]?	What type of software (API, Desktop application etc.) is it?	1			
4	Is the [CE1] [CE2]?	Is the software open source development?	2			
5	What [CE] has the [NM] [DP]?	What pizza has the lowest price?	2	data.	num.	
6	What is the [NM] [CE1] to [OPE] [CE2]?	What is the best/fastest/most robust software to read/edit this data?	3	both	num.	
7	Where do I [OPE] [CE]?	Where do I get updates?	2	obj.		spa.
8	Which are [CE]?	Which are gluten free bases?	1			
9	When did/was [CE] [PE]?	When was the 1.0 version released?	2	data.		tem.
10	What [CE1] do I need to [OPE] [CE2]?	What hardware do I need to run this software?	3	obj.		
11	Which [CE1] [OPE] [QM] [CE2]?	Which pizza has the most toppings?	2	obj.	quan.	
12	Do [CE1] have [QM] values of [DP]?	Do pizzas have different values of size?	2	data.	quan.	

PA = Predicate Arity
RT = Relation Type
M = Modifier
DE = Domain-independent Element
obj. = object property relation, data. = datatype property relation, num. = numeric modifier, quan. = quantitative modifier, tem. = temporal element, spa. = spatial element;
CE = class expression, OPE = object property expression, DP = datatype property, I = individual, NM = numeric modifier, PE = property expression, QM = quantity modifier

Table A.2.: Authoring Tests from [47]

AT	**Parameter**	**Checking**
Occurrence	[E]	E in ontology vocabulary
Class Satisfiability	[CE]	CE is satisfiable
Relation Satisfiability	[CE1][P][E2]	CE1 $\exists P.E2$ is satisfiable, CE1 $\neg\exists P.E2$ is satisfiable
Meta-Instance	[E1][E2]	E1 has type E2
Cardinality Satisfiability	[CE1][n][P][E2]	CE1 $= nP.E2$ is satisfiable, CE1 $\neg = nP.E2$ is satisfiable
Multiple Cardinality (on superlative quantity modifier)	[CE1][P][E2]	$\forall n \geq 0$, CE1 $\neg = nP.E2$ is satisfiable
Comparative Cardinality (on quantity modifier)	[CE1][P1][P2][E1][E2]	$\exists n \geq 0$, CE1 $\leq nP1.E1$ and CE1 $\geq n+1\ P2.E2$ are satisfiable, $\exists m \geq 0$, CE1 $\leq mP2.E2$ and CE2 $\geq (m+1)\ P1.E1$ are satisfiable
Multiple Value (on superlative numeric modifier)	[CE1][P]	$\forall D \subseteq \mathrm{range}(P)$, CE1 $\neg\exists P.D$ is satisfiable
Range	[P]	$\forall P.E$

$\sqcap$ means conjunction
$\neg$ means negation,
$\exists P.E$ means having P relation to some E
= $nP.E$ ($\geq nP.E, \leq nP.E$) means having P relation(s) to exactly (at least, at most) nE
$\forall P.E$ means having P relation (if any) to only E
$\top$ means everything

Table A.3.: Competency Questions (CQs)

Question	**Type**	**Arity**	**Arch**[1]
Core Triage (5Ws & How)			
Are there any detection events on endpoint X?	Binary & Selection	Binary	1
How many {info, low, medium, high} detection events on endpoint X?	Counting	Binary	1

[1] Archetype

Table A.3: continued from previous page

Question	Type	Arity	Arch[1]
Which users are associated with suspicious events on endpoint X?	Selection	Binary	1
Which suspicious processes were executed on endpoint X?	Selection	Binary	1
When did the first detection occur on endpoint X?	Selection	Binary	9
Where did detection X originate?	Selection	Unary	7
How is a detection linked to supporting forensic artifacts?	Selection	Binary	1
Cross-tool and correlation			
Which events from different forensic tools refer to the same underlying activity X?	Selection	Binary	1
How many events from different forensic tools refer to the same underlying activity X?	Counting	Binary	1
Which detections from different tools are correlated to the artifact X?	Selection	Binary	1
Which artifacts share a common endpoint, user, or timestamp window?	Selection	Unary	1
Attack semantics			
Which attack tactic is associated with an artifact?	Selection	Unary	1
Which attack technique is associated with an artifact?	Selection	Unary	1
Which detections are associated with attack tactic X?	Selection	Binary	1
Which detections are associated with attack technique X?	Selection	Binary	1
How many attack techniques and tactics are present?	Counting	Unary	2
Which artifacts support the classification of attack technique X?	Selection	Binary	1
Which endpoints show evidence of lateral movement-related techniques?	Selection	Unary	8
Which suspicious activities are mapped to the same tactic but different techniques?	Selection	Binary	1
Temporal and Lineage CQs			

Table A.3: continued from previous page

Question	Type	Arity	Arch[1]
Which events occurred before and after a specific detection on an endpoint?	Selection	Binary	1
Which process chain led to the execution of a flagged binary?	Selection	Binary	1
Which session events form a coherent attack timeline?	Selection	Unary	8
Is there a specific timeframe with extraneous activities that could be linked to suspicious activities?	Binary	Unary	1
How many artifacts are present for each timeframe?	Counting	Unary	2
What are the artifacts inside a specific timeframe?	Selection	Binary	1
Provenance and traceability			
From which source tool and source file was each normalized entity derived?	Selection	Binary	1
Which original raw records justify a given normalized detection?	Selection	Binary	1
Quality and robustness			
Which entities have incomplete critical attributes (i.e. missing timestamp, endpoint, or user)?	Selection	Binary	1
Which correlated entities contain conflicting values across tools?	Selection	Binary	1

Table A.4.: Competency Questions and Authoring Tests

Competency question	Key elements	Authoring tests
Core triage (5Ws & How)		
Are there any detection events on endpoint X?	DetectionEvent, Endpoint, occursOn	OCC DetectionEvent, Endpoint, occursOn META occursOn : ObjectProperty SAT DetectionEvent is satisfiable

Table A.4: continued from previous page

Competency question	Key elements	Authoring tests
How many {info, low, medium, high} detection events on endpoint X?	DetectionEvent, Level, Endpoint, hasLevel, occursOn	OCC DetectionEvent, Level, Endpoint, hasLevel, occursOn META hasLevel, occursOn : ObjectProperty REL DetectionEvent ⊓ ∃hasLevel.Level satisfiable REL DetectionEvent ⊓ ¬∃hasLevel.Level satisfiable
Which users are associated with suspicious events on endpoint X?	User, SuspiciousEvent, Endpoint, associatedWith, occursOn	OCC User, SuspiciousEvent, Endpoint, associatedWith, occursOn META associatedWith, occursOn : ObjectProperty REL User ⊓ ∃associatedWith.SuspiciousEvent satisfiable REL User ⊓ ¬∃associatedWith.SuspiciousEvent satisfiable
Which suspicious processes were executed on endpoint X?	SuspiciousProcess, Endpoint, executedOn	OCC SuspiciousProcess, Endpoint, executedOn META executedOn : ObjectProperty REL SuspiciousProcess ⊓ ∃executedOn.Endpoint satisfiable REL SuspiciousProcess ⊓ ¬∃executedOn.Endpoint satisfiable

Table A.4: continued from previous page

Competency question	Key elements	Authoring tests
When did the first detection occur on endpoint X?	Detection, Endpoint, timestamp, occursOn	OCC Detection, Endpoint, timestamp, occursOn META timestamp : DatatypeProperty; occursOn : ObjectProperty RNG range(timestamp) $\subseteq$ xsd:dateTime CARD $\forall n \geq 0$, Detection $\sqcap \neg(= n\text{timestamp}.\top)$ satisfiable
Where did detection X originate?	Detection, originatesFrom	OCC Detection, originatesFrom META originatesFrom : ObjectProperty SAT Detection is satisfiable
How is a detection linked to supporting forensic artifacts?	Detection, ForensicArtifact, linkedTo	OCC Detection, ForensicArtifact, linkedTo META linkedTo : ObjectProperty REL Detection $\sqcap$ $\exists$linkedTo.ForensicArtifact satisfiable REL Detection $\sqcap$ $\neg\exists$linkedTo.ForensicArtifact satisfiable
Cross-tool and correlation		
Which events from different forensic tools refer to the same underlying activity X?	Event, ForensicTool, Activity, generatedBy, refersTo	OCC Event, ForensicTool, Activity, generatedBy, refersTo META generatedBy, refersTo : ObjectProperty

Table A.4: continued from previous page

Competency question	Key elements	Authoring tests
		REL Event ⊓ ∃refersTo.Activity satisfiable REL Event ⊓ ¬∃refersTo.Activity satisfiable
How many events from different forensic tools refer to the same underlying activity X?	Event, ForensicTool, Activity, generatedBy, refersTo	OCC Event, ForensicTool, Activity, generatedBy, refersTo REL Event ⊓ ∃refersTo.Activity satisfiable REL Event ⊓ ¬∃refersTo.Activity satisfiable
Which detections from different tools are correlated to the artifact X?	Detection, ForensicTool, Artifact, correlatedTo, generatedBy	OCC Detection, ForensicTool, Artifact, correlatedTo, generatedBy META correlatedTo, generatedBy : ObjectProperty REL Detection ⊓ ∃correlatedTo.Artifact satisfiable REL Detection ⊓ ¬∃correlatedTo.Artifact satisfiable
Which artifacts share a common endpoint, user, or timestamp window?	Artifact, Endpoint, User, associatedEndpoint, associatedUser	OCC Artifact, Endpoint, User, associatedEndpoint, associatedUser META associatedEndpoint, associatedUser : ObjectProperty

Table A.4: continued from previous page

Competency question	Key elements	Authoring tests
		SAT Artifact is satisfiable
Attack semantics		
Which attack tactic is associated with an artifact?	Artifact, AttackTactic, hasTactic	OCC Artifact, AttackTactic, hasTactic META hasTactic : ObjectProperty REL Artifact ⊓ ∃hasTactic.AttackTactic satisfiable REL Artifact ⊓ ¬∃hasTactic.AttackTactic satisfiable
Which attack technique is associated with an artifact?	Artifact, AttackTechnique, hasTechnique	OCC Artifact, AttackTechnique, hasTechnique META hasTechnique : ObjectProperty REL Artifact ⊓ ∃hasTechnique.AttackTechnique satisfiable REL Artifact ⊓ ¬∃hasTechnique.AttackTechnique satisfiable
Which endpoints show evidence of lateral movement-related techniques?	Endpoint, AttackTechnique, LateralMovement, showsEvidenceOf	OCC Endpoint, AttackTechnique, LateralMovement, showsEvidenceOf META showsEvidenceOf : ObjectProperty SAT LateralMovement ⊑ AttackTechnique satisfiable REL Endpoint ⊓ ∃showsEvidenceOf.LateralMovement satisfiable

Table A.4: continued from previous page

Competency question	Key elements	Authoring tests
Quality and robustness		
Which entities have incomplete critical attributes?	Entity, timestamp, endpoint, user	OCC Entity, timestamp, associatedEndpoint, associatedUser META timestamp : DatatypeProperty RNG range(timestamp) ⊆ xsd:dateTime REL Entity ⊓ ¬∃timestamp.⊤ satisfiable

Appendix B.

Listings

```yaml
# YAML file
common: &common
  - [ex:name, $(name)]
  - [ex:age, $(age)]

po:
  - *common
  - [ex:city, $(city)]

# Expected result:
po:
  - [ex:name, $(name)]
  - [ex:age, $(age)]
  - [ex:city, $(city)]

# Actually
po:
  -
    - [ex:name, $(name)]
    - [ex:age, $(age)]
  - [ex:city, $(city)]
```

Listing B.1: Example of YAML issues with anchors and lists

Bibliography

[1] DeepL SE. *DeepL Write*. AI writing assistant used for grammar and style improvement. URL: https://www.deepl.com/write (Last accessed on 7/4/2026).

[2] Sakshi Singh and Suresh Kumar. "The Times of Cyber Attacks". In: *ACTA Technica Corviniensis – Bulletin of Engineering* 13.3 (July 2020), pp. 133–137. ISSN: 2067-3809. URL: https://acta.fih.upt.ro/pdf/2020-3/ACTA-2020-3-25.pdf (Last accessed on 15/5/2026).

[3] Hiscox. *Hiscox Cyber Readiness Report 2025*. 2025. URL: https://www.hiscoxgroup.com/hiscox-cyber-readiness-report-2025 (Last accessed on 15/5/2026).

[4] Christos Makridis and Benjamin Dean. "Measuring the economic effects of data breaches on firm outcomes: Challenges and opportunities". In: *Journal of Economic and Social Measurement* 43.1-2 (Feb. 2018), pp. 59–83. ISSN: 1875-8932. DOI: 10.3233/jem-180450.

[5] World Economic Forum. *The Global Risks Report 2020*. Insight Report 15th Edition. Jan. 15, 2020. URL: https://www3.weforum.org/docs/WEF_Global_Risk_Report_2020.pdf (Last accessed on 15/5/2026).

[6] Raffael Marty. "Cloud application logging for forensics". In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. SAC'11. ACM, Mar. 2011, pp. 178–184. DOI: 10.1145/1982185.1982226.

[7] Yamato Security. *Hayabusa: A Sigma-based Threat Hunting and Fast Forensics Timeline Generator*. Version 2.17.0. Yamato Security. URL: https://github.com/Yamato-Security/hayabusa (Last accessed on 11/3/2026).

[8] SigmaHQ. *Sigma Specifications*. 2025. URL: https://github.com/SigmaHQ/sigma-specification (Last accessed on 15/5/2026).

[9] WithSecure Labs. *Chainsaw: Rapidly Search and Hunt through Windows Forensic Artefacts*. Version 2.14.0. URL: https://github.com/WithSecureLabs/chainsaw (Last accessed on 11/3/2026).

[10] Eran Salfati and Michael Pease. *Digital Forensics and Incident Response (DFIR) framework for Operational Technology (OT)*. Tech. rep. National Institute of Standards and Technology (U.S.), June 2022. DOI: 10.6028/nist.ir.8428. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934922 (Last accessed on 11/3/2026).

[11] Abdullah Ayub Khan et al. “Digital forensics and cyber forensics investigation: security challenges, limitations, open issues, and future direction”. In: *International Journal of Electronic Security and Digital Forensics* 14.2 (2022), p. 124. ISSN: 1751-9128. DOI: 10.1504/ijesdf.2022.121174.

[12] Gerard Johansen. *Digital Forensics and Incident Response: Incident response tools and techniques for effective cyber threat response*. Third edition. Birmingham: Packt Publishing, 2022. ISBN: 1803230258.

[13] Mohamed Amine Batoun et al. “A literature review and existing challenges on software logging practices: From the creation to the analysis of software logs”. In: *Empirical Software Engineering* 29.4 (2024). ISSN: 1573-7616. DOI: 10.1007/s10664-024-10452-w.

[14] Ilker Kara. “Read the digital fingerprints: log analysis for digital forensics and security”. In: *Computer Fraud & Security* 2021.7 (July 2021), pp. 11–16. ISSN: 1361-3723. DOI: 10.1016/s1361-3723(21)00074-9.

[15] Purvi Pokhariyal, Archana Patel, and Shubham Pandey. *AI and Emerging Technologies: Automated Decision-Making, Digital Forensics, and Ethical Considerations*. CRC Press, Nov. 2024. ISBN: 9781003501152. DOI: 10.1201/9781003501152.

[16] Athanasios Dimitriadis et al. “D4I - Digital forensics framework for reviewing and investigating cyber attacks”. In: *Array* 5 (Mar. 2020), p. 100015. ISSN: 2590-0056. DOI: 10.1016/j.array.2019.100015.

[17] Brian D. Carrier and Eugene H. Spafford. “Getting Physical with the Digital Investigation Process”. In: *Int. J. Digit. EVid.* 2 (2003). URL: https://api.semanticscholar.org/CorpusID:383070.

[18] Hai-Cheng Chu, Der-Jiunn Deng, and Han-Chieh Chao. “An Ontology-driven Model for Digital Forensics Investigations of Computer Incidents under the Ubiquitous Computing Environments”. In: *Wireless Personal Communications* 56.1 (Dec. 2009), pp. 5–19. ISSN: 1572-834X. DOI: 10.1007/s11277-009-9886-x.

[19] Darren Quick and Kim-Kwang Raymond Choo. *Big Digital Forensic Data: Volume 1: Data Reduction Framework and Selective Imaging*. Springer Singapore, 2018. ISBN: 9789811077630. DOI: 10.1007/978-981-10-7763-0.

[20] Hudan Studiawan, Ferdous Sohel, and Christian Payne. “Automatic log parser to support forensic analysis”. In: *Australian Digital Forensics Conference* (2018). DOI: 10.25958/5C5268C766686. URL: https://ro.ecu.edu.au/adf/178/ (Last accessed on 25/2/2026).

[21] Pinjia He et al. “An Evaluation Study on Log Parsing and Its Use in Log Mining”. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, June 2016, pp. 654–661. DOI: 10.1109/dsn.2016.66.

[22] Sarunas Grigaliunas, Jevgenijus Toldinas, and Algimantas Venckauskas. “An Ontology-Based Transformation Model for the Digital Forensics Domain”. In: *Elektronika ir Elektrotechnika* 23.3 (June 2017), pp. 78–82. ISSN: 1392-1215. DOI: 10.5755/j01.eie.23.3.18337.

[23] Yoan Chabot et al. “A complete formalized knowledge representation model for advanced digital forensics timeline analysis”. In: *Digital Investigation* 11 (2014), S95–S105. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.05.009.

[24] Fabrizio Turchi. “Evidence Exchange Under the EIO: Technological Challenges”. In: *European Investigation Order*. Springer International Publishing, 2023, pp. 35–52. ISBN: 9783031316869. DOI: 10.1007/978-3-031-31686-9_5.

[25] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Computing Surveys* 54.4 (July 2021), pp. 1–37. ISSN: 1557-7341. DOI: 10.1145/3447772.

[26] Bradley Schatz, George Mohay, and Andrew Clark. “Rich Event Representation for Computer Forensics”. In: *Proceedings of the Fifth Asia-Pacific Industrial Engineering and Management Systems Conference (APIEMS 2004)*. Ed. by E Kozan. Brisbane, Queensland: Queensland University of Technology Publications, 2004, pp. 1–16. URL: https://eprints.qut.edu.au/23955/ (Last accessed on 15/5/2026).

[27] Yoan Chabot. “Construction, Enrichment and Semantic Analysis of Timelines: Application to Digital Forensics”. PhD thesis. University College Dublin and University of Burgundy., Nov. 2015.

[28] OASIS Cyber Threat Intelligence (CTI) Technical Committee. *STIX Version 2.1*. Tech. rep. OASIS Open, 2022. URL: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.pdf (Last accessed on 15/5/2026).

[29] Luca Cotti et al. *Enabling Transparent Cyber Threat Intelligence Combining Large Language Models and Domain Ontologies*. 2025. DOI: 10.48550/ARXIV.2509.00081. URL: https://arxiv.org/abs/2509.00081 (Last accessed on 7/4/2026).

[30] Eoghan Casey et al. “Advancing coordinated cyber-investigations and tool interoperability using a community developed specification language”. In: *Digital Investigation* 22 (Sept. 2017), pp. 14–45. ISSN: 1742-2876. DOI: 10.1016/j.diin.2017.08.002.

[31] Ágney Lopes Roth Ferraz et al. *The Procedural Semantics Gap in Structured CTI: A Measurement-Driven STIX Analysis for APT Emulation*. 2025. DOI: 10.48550/ARXIV.2512.12078. URL: https://arxiv.org/abs/2512.12078 (Last accessed on 7/4/2026).

[32] Zareen Syed et al. “UCO: A Unified Cybersecurity Ontology”. In: Feb. 2016, AAAI Workshop on Artificial Intelligence for Cyber Security.

[33] CASE Community. *CASE Ontology Design and Specification*. Version 1.0. URL: https://caseontology.org/resources/case_design_document.html (Last accessed on 7/4/2026).

[34] Lionel Tailhardat, Yoan Chabot, and Raphael Troncy. “NORIA-O: An Ontology for Anomaly Detection and Incident Management in ICT Systems”. In: *The Semantic Web*. Springer Nature Switzerland, 2024, pp. 21–39. ISBN: 9783031606359. DOI: 10.1007/978-3-031-60635-9_2.

[35] Orange. *NORIA : détection d’anomalies réseaux à l’aide des graphes de connaissances*. Blog post on knowledge graph-based anomaly detection in network systems. Orange Hello Future. 2024. URL: https://hellofuture.orange.com/fr/noria-detection-danomalies-reseaux-a-laide-des-graphes-de-connaissances/ (Last accessed on 15/5/2026).

[36] Yee Ching Tok, Davis Yang Zheng, and Sudipta Chattopadhyay. “A Smart City Infrastructure ontology for threats, cybercrime, and digital forensic investigation”. In: *Forensic Science International: Digital Investigation* 52 (Mar. 2025), p. 301883. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2025.301883.

[37] Eric Xu, Wenbin Zhang, and Weifeng Xu. “Transforming Digital Forensics with Large Language Models: Unlocking Automation, Insights, and Justice”. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. CIKM ’24. ACM, Oct. 2024, pp. 5543–5546. DOI: 10.1145/3627673.3679091.

[38] Akansha Shukla et al. *RuleGenie: SIEM Detection Rule Set Optimization*. 2025. DOI: 10.48550/ARXIV.2505.06701. URL: https://arxiv.org/abs/2505.06701 (Last accessed on 7/4/2026).

[39] Jeel Piyushkumar Khatiwala, Daniel Kwaku Ntiamoah Addai, and Weifeng Xu. “Evaluating the Reliability of Digital Forensic Evidence Discovered by Large Language Model: A Case Study”. In: *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, July 2025, pp. 1067–1076. DOI: 10.1109/compsac65507.2025.00138.

[40] Seyed Reza Nabavi, Zhiyuan Wang, and Gade Pandu Rangaiah. “Sensitivity Analysis of Multi-Criteria Decision-Making Methods for Engineering Applications”. In: *Industrial & Engineering Chemistry Research* 62 (2023), pp. 6707–6722. DOI: 10.1021/acs.iecr.2c04270.

[41] Lauren Hyun Ee Wong et al. “A Comparative Study of Relational, Graph, Wide Column, Key-Value, and Document Models in Retail Industries”. In: *Information Research Communications* 2.1 (Sept. 2025), pp. 71–97. ISSN: 3041-0800. DOI: 10.5530/irc.2.1.6.

[42] Renzo Angles and Claudio Gutierrez. “An Introduction to Graph Data Management”. In: *Graph Data Management*. Springer International Publishing, 2018, pp. 1–32. ISBN: 9783319961934. DOI: 10.1007/978-3-319-96193-4_1.

[43] Michael Kaufmann and Andreas Meier. *SQL and NoSQL Databases: Modeling, Languages, Security and Architectures for Big Data Management*. Springer Nature Switzerland, 2023. ISBN: 9783031279089. DOI: 10.1007/978-3-031-27908-9.

[44] Ronald Denaux et al. “Knowledge Architecture for Organisations”. In: *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer International Publishing, 2017, pp. 57–84. ISBN: 9783319456546. DOI: 10.1007/978-3-319-45654-6_3.

[45] Ricardo de Almeida Falbo. “SABiO: Systematic approach for building ontologies”. In: *CEUR Workshop Proceedings* 1301 (Jan. 2014).

[46] Mike Uschold and Michael Gruninger. “Ontologies: principles, methods and applications”. In: *The Knowledge Engineering Review* 11.2 (1996), pp. 93–136. ISSN: 1469-8005. DOI: 10.1017/s0269888900007797.

[47] Yuan Ren et al. “Towards Competency Question-Driven Ontology Authoring”. In: *The Semantic Web: Trends and Challenges*. Springer International Publishing, 2014, pp. 752–767. ISBN: 9783319074436. DOI: 10.1007/978-3-319-07443-6_50.

[48] Youssra Rebboud et al. “Can LLMs Generate Competency Questions?” In: *The Semantic Web: ESWC 2024 Satellite Events*. Springer Nature Switzerland, 2025, pp. 71–80. ISBN: 9783031789526. DOI: 10.1007/978-3-031-78952-6_7.

[49] Xi Ye and Greg Durrett. “The unreliability of explanations in few-shot prompting for textual reasoning”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. 2022. ISBN: 9781713871088.

[50] Mark A. Musen. “The protégé project: a look back and a look forward”. In: *AI Matters* 1.4 (2015), pp. 4–12. ISSN: 2372-3483. DOI: 10.1145/2757001.2757003.

[51] Jeff Z. Pan. “Resource Description Framework”. In: *Handbook on Ontologies*. Springer Berlin Heidelberg, 2009, pp. 71–90. ISBN: 9783540926733. DOI: 10.1007/978-3-540-92673-3_3.

[52] Marlon A. Altamirano Di Luca and Neilys González Benítez. “Comparative Study of RDF and OWL Ontology Languages as Support for the Semantic Web”. In: *Applied Technologies*. Springer International Publishing, 2020, pp. 3–12. ISBN: 9783030425173. DOI: 10.1007/978-3-030-42517-3_1.

[53] Diana Kalibatiene and Olegas Vasilecas. “Survey on Ontology Languages”. In: *Perspectives in Business Informatics Research*. Springer Berlin Heidelberg, 2011, pp. 124–141. ISBN: 9783642245114. DOI: 10.1007/978-3-642-24511-4_10.

[54] Dave Reynolds et al. *An assessment of RDF/OWL modelling*. Tech. rep. HPL-2005-189. Hewlett-Packard Laboratories, 2005. URL: https://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2005/HPL-2005-189.html (Last accessed on 6/5/2026).

[55] VisualDataWeb. *WebVOWL: Visualizing Ontologies on the Web*. Version 1.1.6. 2026. URL: https://github.com/VisualDataWeb/WebVOWL (Last accessed on 5/6/2026).

[56] Daniel Garijo. “WIDOCO: a wizard for documenting ontologies”. In: *International Semantic Web Conference*. Springer, Cham. 2017, pp. 94–102. DOI: 10.1007/978-3-319-68204-4_9. URL: http://dgarijo.com/papers/widoco-iswc2017.pdf.

[57] Louan Robert. *UniForeXT*. Github repository of this master thesis. 2026. URL: https://github.com/louanrobert/UniForeXT (Last accessed on 26/5/2026).

[58] Louan Robert. *louanrobert/UniForeXT*. 2026. DOI: 10.5281/ZENODO.20396206. URL: https://zenodo.org/doi/10.5281/zenodo.20396206 (Last accessed on 27/5/2026).

[59] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. “OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 10.2 (2014), pp. 7–34. DOI: 10.4018/ijswis.2014040102.

[60] Paolo Pareti and George Konstantinidis. “A Review of SHACL: From Data Validation to Schema Reasoning for RDF Graphs”. In: *Reasoning Web. Declarative Artificial Intelligence*. Springer International Publishing, 2022, pp. 115–144. ISBN: 9783030954819. DOI: 10.1007/978-3-030-95481-9_6.

[61] Andrea Cimmino, Alba Fernández-Izquierdo, and Raúl García-Castro. “Astrea: Automatic Generation of SHACL Shapes from Ontologies”. In: *The Semantic Web*. Springer International Publishing, 2020, pp. 497–513. ISBN: 9783030494612. DOI: 10.1007/978-3-030-49461-2_29.

[62] Thomas Francart. *SHACL Play!* Version 0.12.0. Sparna, 2019. URL: https://github.com/sparna-git/shacl-play (Last accessed on 18/5/2026).

[63] B. McBride. “Jena: a semantic Web toolkit”. In: *IEEE Internet Computing* 6.6 (2002), pp. 55–59. ISSN: 1089-7801. DOI: 10.1109/mic.2002.1067737.

[64] Anastasia Dimou et al. “RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data”. In: *Proceedings of the 7th Workshop on Linked Data on the Web*. Ed. by Christian Bizer et al. Vol. 1184. CEUR Workshop Proceedings. CEUR, 2014.

[65] Apostolos Ampatzoglou et al. “Applying the Single Responsibility Principle in Industry: Modularity Benefits and Trade-offs”. In: *Proceedings of the Evaluation and Assessment on Software Engineering*. EASE ’19. ACM, 2019, pp. 347–352. DOI: 10.1145/3319008.3320125.

[66] Brahma Dathan and Sarnath Ramnath. *Object-Oriented Analysis, Design and Implementation: An Integrated Approach*. Springer Nature Switzerland, 2025. ISBN: 9783031712401. DOI: 10.1007/978-3-031-71240-1.

[67] Olaf Musch. *Design Patterns with Java: An Introduction*. Springer Fachmedien Wiesbaden, 2023. ISBN: 9783658398293. DOI: 10.1007/978-3-658-39829-3.

[68] OpenJDK Community. *OpenJFX*. Open-source JavaFX platform. URL: https://openjfx.io/ (Last accessed on 24/4/2026).

[69] Chart.js Contributors. *Chart.js*. URL: https://www.chartjs.org/ (Last accessed on 24/4/2026).

[70] vis.js Community. *vis.js*. URL: https://visjs.org/ (Last accessed on 24/4/2026).

[71] Axel Polleres. “SPARQL”. In: *Encyclopedia of Social Network Analysis and Mining*. Springer New York, 2014, pp. 1960–1966. ISBN: 9781461461708. DOI: 10.1007/978-1-4614-6170-8_124.