CAMILLA: Contextual Autopsy Mapping Integrated Linking Layer

Daniele D'Agnelli
School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK

ABSTRACT

Context
Digital Forensics (DF) is the science of applying technical and investigative skills in service of the law, using a process that recovers, reveals, relates, and reconstructs actions from digital devices in criminal and civil court cases.

Aim
This Master Thesis aims to demonstrate that it is possible to simplify and standardise generating visualisations to support the DF reporting process by implementing a plugin fully integrated into the well-known Open Source platform Autopsy.

Method
A plugin is developed for the Autopsy software to allow users to interact with the artefacts discovered with the objective of facilitating the creation of standardised visualisations to enrich DF reports.

Keywords
digital forensics, reporting, visualisations, graph, entity-relationship

1. INTRODUCTION
Generally, format and quality of delivery impact credibility and can create compelling narratives. The software being developed as part of this thesis aims to minimise this class of non-legal variables and enable fairer outcomes, minimising social inequality and inequality of punishment, by releasing this for an Open Source software. This thesis attempts to build a software plugin for the well-known Open Source DF platform Autopsy called **CAMILLA: Contextual Autopsy Mapping Integrated Linking Layer**.

2. BACKGROUND
Multiple studies have attempted to establish if there is a correlation between criminal justice outcomes and non-legal factors. In the research paper "*Public versus private defence: Can money buy justice?*", it was found that, in the UK, there are comparable outcomes between privately and publicly defended accused individuals, suggesting that justice cannot be bought (Rattner, A., Turjeman, H., and Fishman, G., 2008). At the same time, it has been observed that forensic evidence impacts many of the decision points in the justice process, with the availability of forensic evidence linked to an increased likelihood of arrest and length of the sentence (Peterson, J.L. et al., 2012). Due to the nature of the investigation process and the lack of a standard way to generate those reports, there is a degree of freedom for a DF professional to present their results and explanation of what they believe happened. It is, therefore, worth researching whether the quality of these forensic reports can impact these outcomes.

While it is well-established that marginalised groups worldwide are overrepresented in a given society's prison population, research has not yet established whether there is a link between social inequality and criminal justice outcomes. Therefore, it is not yet known if social inequality can translate into inequality of punishment (Karstedt, S. 2021). Multiple studies have attempted to establish if there is a correlation between criminal justice outcomes and non-legal factors. In the research paper "*Public versus private defence: Can money buy justice?*", it was found that, in the UK, there are comparable outcomes between privately and publicly defended accused individuals, suggesting that justice cannot be bought (Rattner, A., Turjeman, H., and Fishman, G., 2008). At the same time, it has been observed that forensic evidence impacts many of the decision points in the justice process, with the availability of forensic evidence linked to an increased likelihood of arrest and length of the sentence (Peterson, J.L. et al., 2012). Due to the nature of the investigation process and the lack of a standard way to generate those reports, there is a degree of freedom for a DF professional to present their results and explanation of what they believe happened.

Therefore, it is worth developing Open Source software to give access to a large population of DF professionals the necessary tools to create effective, standardised visualisations to enrich their reports.

2.1 The Investigative Process
The investigative process, also used in DF, can be broken down into the following seven steps:

1. Identification
2. Acquisition
3. Preservation
4. Search
5. Analysis
6. Reconstruction
7. Presentation

The last step of the process is the most subjective: presentation, which includes reporting, for which there currently is no required standard to be met. This means that the perceived quality and richness of the work can be biased towards more compelling narratives when elaborating and presenting the findings. DF reports can be complemented by visual representation of relationships between subjects, with two of the most common approaches involving using graphs (**Figure 1**) or showing events in timelines (**Figure 2**).

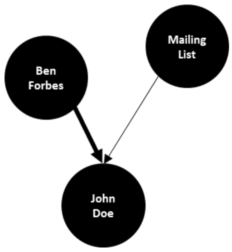

Figure 1 Example of Graph from the Don Joe Case

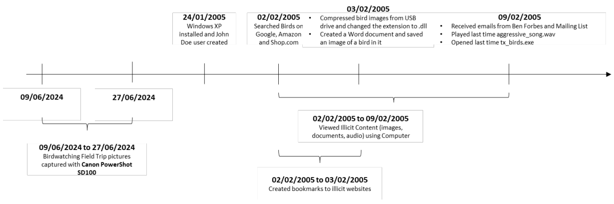

Figure 2 Example of Timeline from the Don Joe Case

These visualisations and other non-material elements of the final report could be non-legal elements that can influence the court toward a specific interpretation of how the events occurred. CAMILLA attempts to mitigate this risk by simplifying and standardising the creation of these artefacts.

2.2 Existing work

Data visualisation occupies a vast segment of the market. Numerous standalone software solutions allow users to create visual representations of data, commonly falling under the banner of Business Intelligence. Conversely, software options dedicated to managing DF cases are limited. Prominent examples include the commercial software EnCase and the Open Source software Autopsy. There is a market gap at the juncture of DF case management and data visualisation, wherein an analyst can interactively visualise and manipulate artefacts within a case. This thesis primarily addresses the Open Source software Autopsy. Although Autopsy supports extensions via plugins implemented in either Java or Python, only Java plugins can fully access and extend both the Autopsy data model and its UI. These plugins are inherently more intricate to develop. While the Autopsy website offers some documentation for the Java API, the lack of comprehensively documented examples presents a significant barrier for potential developers. Moreover, no other Open-Source plugins like CAMILLA currently aim to organise and visualise Autopsy cases, and the research couldn't find any examples of commercial ones either.

2.3 Autopsy and The Sleuth Toolkit

Autopsy and The Sleuth Kit (TSK) are two widely used tools to perform Digital Forensics and facilitate the examination of digital devices. The Sleuth Kit (TSK) is an Open Source library that enables the non-intrusive analysis of various file systems, preserving the integrity of the original data.

Autopsy is a DF platform developed using the Java programming language and the NetBeans application framework that leverages TSK to support the investigative process. The latest version of Autopsy is 4, distributed under the Apache 2 License, an Open Source license that explicitly permits free use, modification, and distribution of the software. This licensing model aligns with the Open Source philosophy and ensures the software's availability to a broad audience.

3. Objectives

This thesis attempts to establish the feasibility of developing an Autopsy plugin that enables professionals to visualise artefacts found during the DF investigation process to enrich the final report. This thesis attempts to demonstrate that it is possible to develop a professional product, using Open Source technologies, with a level of quality that makes it readily usable for the wide audience it is intended to. This project leverages Software Engineering principles of specifying requirements to develop a Minimum Viable Product (MVP). Software is considered MVP when it works in a real-life setting and provides value to its target audience. Whilst in a Proof-of-content (POC), the only point is validating the feasibility of development, an MVP sets the bar above that threshold in releasing software fit for purpose in a production scenario.

3.1 Requirements

The current state of development addresses the following requirements, which have been prioritised based on the impact and the value they can provide to the investigative process. Requirements have been expressed by borrowing from the Agile methodology the technique of specifying them through User Stories and Acceptance Criteria.

#1 User Story: As a user, I want to visualise the relationships existing among the relevant artefacts I identified using Autopsy.

Acceptance Criteria:

- It is possible to drag any Autopsy artefact onto a canvas and represent it through an icon.
- It is possible to visually link multiple artefacts to express that they are in a relationship.
- It is possible to move artefacts on the canvas and organise them spacially as desired.
- It is possible to delete an artefact (edge) or a relationship (vertex) on the canvas.
- It is possible to resize the icon associated with an edge.
- It is possible to select multiple vertexes and edges simultaneously.
- It is possible to delete the selected vertex and edges.
- It is not possible to rename a vertex.

#2 User Story: As a user, I want to annotate my entity graph visualisation

Acceptance Criteria:

- It is possible to label an edge to add details in natural language on the nature of the relationship.
- It is possible to create a note box by right-clicking on a blank part of the canvas.
- It is possible to add a vertex between a note box and any artefacts.

#3 User Story: As a user, I want to export the visualisation so that it can be added to my report.

Acceptance Criteria:

- It is possible to download the visualisation on screen in an image format by pressing a button.
- It is possible for the user to specify the name of the file to be saved.

- The extension of the image file is automatically added if missing.
- The saved exported image is automatically opened for preview upon saving.

#3 User Story: As a user, I want the visualisation of the relationship to be saved along with the Autopsy case so that I can keep working on it upon reopening the case
Acceptance Criteria:
- The visualisation is saved automatically at every change in the embedded SQLite database.
- If a visualisation exists for a case, it is automatically opened along with the case itself.
- It is possible to continue working on a loaded visualisation

4. Autopsy Architecture

Autopsy is a complex platform developed in Java that leverages NetBeans and its design patterns for robustness. Developing a Java Plugin for Autopsy requires an understanding of all the used components and how they interact with each other. **Figure 3** introduces the Autopsy Architecture as well as the components and dependencies required for the implementation of CAMILLA.

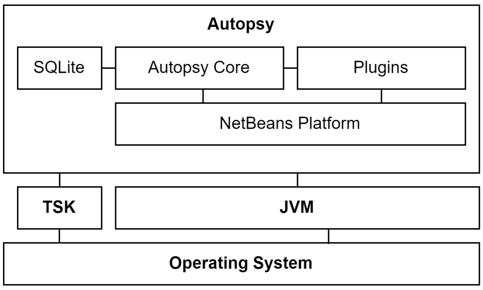

Figure 3 Autopsy Architecture

The different components shown in **Figure 3** are defined as follows:

Operating System: Autopsy can be executed on any Operating System that can run a JVM, with Windows, macOS and GNU/Linux being the most common. CAMILLA has been developed and tested only on Microsoft Windows and may not work as expected on other platforms out of the box.

JVM: Provide an abstraction layer over the OS to improve cross-platform compatibility. CAMILLA has been written using OpenJDK 1.8.

TSK: The Sleuth Kit is a set of standalone command-line utilities written in C to interact with file images and filesystems. Autopsy leverages these utilities and exposes them through a Desktop-based GUI.

NetBeans Platform: It's the infrastructure on the top of which Autopsy is built. NetBeans is both the IDE used to develop Autopsy as well as the libraries that provide the backbone of the app itself and the patterns to be used to extend any application that leverages it. Most notably, NetBeans facilitates the creation of the UI by allowing the registration of components using annotation in the Java Code and a mechanism to register the events to which the application should react.

Autopsy Core: Following the design patterns offered by NetBeans, Autopsy is organised in modules. The main module is called Autopsy Core, and it provides most of the functionalities that can be seen when installing the application.

Plugins: Autopsy is a modular architecture that supports extensions of its capabilities through Plugins. There are two kinds of plugins that can be developed:
- Python Plugins
- Java Plugins

Python plugins represent a lightweight form of extending Autopsy. The Python plugins are mainly used during the ingestion process. It is impossible to modify the Autopsy UI using a Python plugin or access the case's backend features.
On the other end, Java plugins are first-class citizens in the Autopsy architecture and can access all the public APIs made available by Autopsy.
An Autopsy Java plugin is a NetBeans Module (NBM), compiled and distributed as an individual file with **.nbm** extension and loaded in the main application at runtime through the Plugin functionality Autopsy inherits from NetBeans. CAMILLA has been created using NetBeans by creating a new project as NetBeans Module, as shown in **Figure 4**.

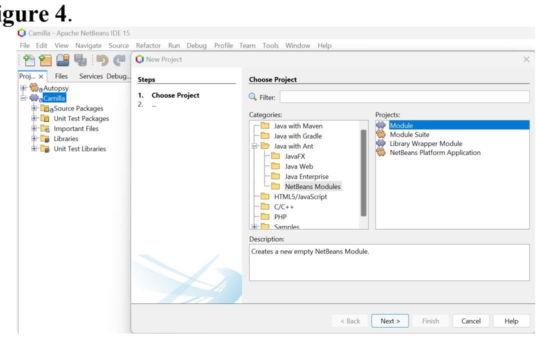

Figure 4 CAMILLA as a Module in NetBeans.

Java plugins can be of one of the four following types:
- **Ingest**: analyse data from a data source
- **Report:** create an export report
- **Content Viewer:** show the content of one artefact
- **Result Viewer:** shows multiple artefacts

While Ingest and Report models do not have a UI component, Content and Result Viewer do and are rendered as specified by Autopsy, as highlighted in
Figure 5. All the components rendered on screen are NetBeans components, and in particular, they are subclasses of ***TopComponent***, the basic unit of display offered in a NetBeans app: when a Module implements a class that extends ***TopComponent*** it is displayed as part of the app.

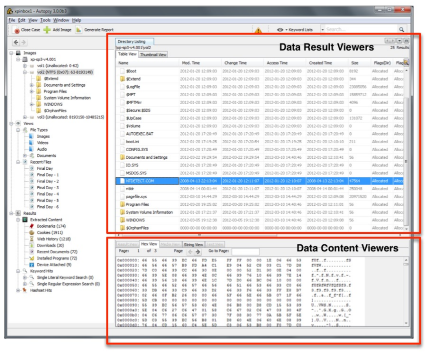

Figure 5 Modules and GUI layout in Autopsy

SQLite: Persistence is offered by Autopsy through a library that implements SQLite, a popular Open Source file-based RDBMS.

5. CAMILLA

This chapter delves into the detailed construction of the plugin, exploring the design patterns and overarching architecture. It also provides insights into the design decisions and the reasoning behind them.

5.1 Code Organisation

The code for CAMILLA is organised into Java packages to simplify comprehension and expansion of the overall project by effectively leveraging the namespace. With the project's scope and future scalability in mind, two distinct packages have been employed, as shown in **Figure 6**. The root namespace for the project is **net.bidimensional.camilla**, adhering to the standard Java naming convention that utilises a public domain name written in reverse. This approach minimises the likelihood of namespace conflicts when the package is imported into other projects. The domain 'bidimensional.net' in CAMILLA's root corresponds to the personal domain name of this document's author.

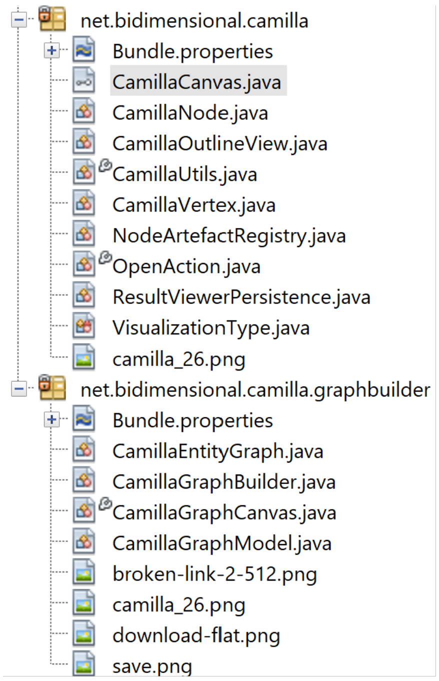

Figure 6 CAMILLA Packages structure

5.1.1 net.bidimensional.camilla

The root of the project encompasses classes that are sufficiently generic, such that their applicability is not confined solely to the visualisation of entities in graph format.

- **Bundle.properties** serves as a centralised datastore containing labels that can be consistently referenced across the codebase. This architectural approach, facilitated by NetBeans through the NbBundle class, proves particularly invaluable in localisation scenarios. It ensures all textual strings are collated in a single location and are accessed via a designated variable name.
- **CamillaCanvas.java** implements an interface prescribing a single method, **getGraphComponent().** This design abstraction generalises the notion of a drawing canvas while obfuscating its underlying implementation using JGraphX.
- **CamillaOutlineView.java** extends the NetBeans object *OutlineView,* the Autopsy artefacts' UI component is dragged from. This class overrides some inherited methods that implement the transfer of data when the component is drag-and-dropped from the UI.
  - **createTransferable:** In the auxiliary class CamillaTransferHandler, the createTransferable method generates a **Transferable** object, representing data suitable for drag-and-drop operations in the GUI. It determines if the component in question is an instance of the Outline class. Then, it identifies in the Outline the object being selected and wraps it into a NodeTransferable object, which is then returned.
  - **getSourceActions:** Within the confines of the CamillaTransferHandler auxiliary class, this method exposes the types of operations, such as copy or move, that a component can undertake during data transfer. By returning the COPY_OR_MOVE constant, the method indicates dual functionality. Specifically in NetBeans, the COPY_OR_MOVE flag denotes that data from the source component can be either copied or moved during a drag-and-drop operation, which allows great flexibility in what can be done with the transferred data.
  - **getTransferDataFlavors:** This method is situated in the private NodeTransferable class, and defines the nature of the data being transferred via DataFlavor objects. It stipulates that the data for transfer is precisely of the Node type. In the NetBeans context, DataFlavor is a crucial mechanism to describe the type or kind of data that's being dragged, ensuring data compatibility between the source and target components.
  - **isDataFlavorSupported:** Also housed within the NodeTransferable class, the isDataFlavorSupported method checks the compatibility of the provided DataFlavor with the predefined flavours. If there's a match with the Node class representation, it returns true and false otherwise. In NetBeans, this method acts as a safeguard, ensuring that data being transferred (or dropped) matches the expected type or format, thus preventing potential data type conflicts and ensuring smooth data operations.

This is one of the most important classes implemented as part of this project, as it implements the core capability necessary to visually manipulate objects on the canvas.

- **CamillaUtils.java** is a utility class. It is discussed in more detail in paragraph 5.5.
- **CamillaVertex.java** encapsulates a singular attribute: a Node object sourced from the NetBeans platform's org.openide.nodes.Node. To facilitate external interactions, this class exposes the getNode() method, granting read-only access to its internal Node attribute. Moreover, the overridden toString() method yields the display name of the encapsulated Node, aligning the string representation of a CamillaVertex instance with the semantics of the underlying Node. The CamillaVertex class acts as an intermediary, assimilating the NetBeans platform's node framework into a tailored representation suitable for the visualisation on the Canvas.
- **NodeArtefactRegistry.java** serves as a mediator that manages a mapping between instances of

BlackboardArtifact and Node. Specifically, this class leverages the Java Collections Framework, utilising a HashMap to maintain these relationships, with BlackboardArtifact instances as keys and corresponding Node instances as values. This class has been developed to lookup artefacts in the Outline for load operations but is not currently used.

- **OpenAction.java** is responsible for the rendering of the CAMILLA icon on the Autopsy toolbar and the associated action that shows the About dialog. This class is derived from CallableSystemAction, a unique type of action made available by the NetBeans Platform. The action is registered through the @ActionRegistration annotation, ensuring that it is integrated within the Autopsy GUI and responds under the label "Camilla.". Several key attributes underline the design of this class: the action is marked by the identifier "net.bidimensional.camilla.OpenAction" and is categorised under the "Tools" section. Annotations such as @ActionReferences determine the action's location, positioning it both within the "Tools" menu (**Figure 7)** and a dedicated toolbar, as shown in **Figure 8** .

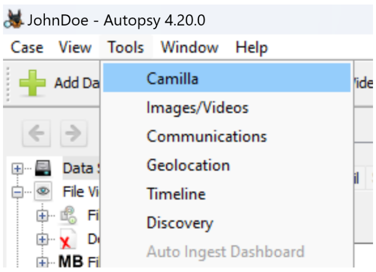

Figure 7 CAMILLA toolbar Menu item

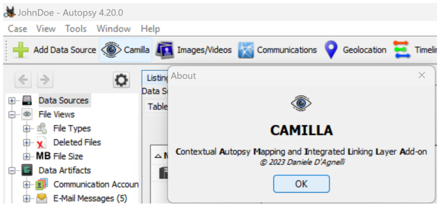

Figure 8 CAMILLA toolbar icon and About dialog

In its lifecycle, the OpenAction constructor initialises both the toolbar button and menu item to trigger the associated action. The menu item and the toolbar icon both call the ***performAction*** method. When invoked, this method displays an informational dialog showcasing both the Camilla logo and a brief descriptive acronym for "CAMILLA,".

- **ResultViewerPersistance.java** facilitates the management of user preferences related to table display within an application. This class is a fork of a class with the same name from Autopsy, which has been imported into the project as some of the required methods didn't have the necessary scope visibility and, therefore, couldn't be used by CAMILLA.
- **VisualizationType.java** implements an Enum that can take one of two possible types: ENTITY or TIMELINE. This class was designed for extensibility, allowing for the management of various visualisation types in the future. The primary purpose of this enumerator is to determine the actions based on the visualisation type. Currently, only actions corresponding to the TIMELINE value are implemented.

5.1.2 net.bidimensional.camilla.graphbuilder

This package holds all classes specifically tailored for graph-building capabilities. Should the CAMILLA plugin undergo expansion and implement features such as a timeline builder, it would be logical to instantiate a sibling package.

- **Bundle.properties** is analogous to the homonymous file in the root package of the project, but its variable has a visibility restricted visibility to the graphbuilder package.
- **CamillaEntityGraph.java** is a wrapper class that extends JGraphx's mxGraph, essentially serving as a renamed version of the latter. This approach was adopted to ensure future scalability and to minimise the direct reliance of CAMILLA's implementation on a specific library.
- **CamillaGraphBuilder.java** is the primary class responsible for the content displayed on the screen and serves as the central component of the entire plugin. Section 5.4 provides an in-depth exploration of this class and its relationship with the broader Autopsy ecosystem.
- **CamillaCanvas.java** is the visible part of the UI that serves as a canvas for the dropped vertexes. This class is also explained in more detail in section 5.4.2.
- **CamillaGraphModel.java** functions as a wrapper for the JGraphX mxGraph class. Its sole objective is to facilitate the decoupling of CAMILLA's implementation from its dependency on the JGraphX library.

5.2 Plugin Registration

A **Result Viewer** Autopsy module is required to visualise multiple types of artefacts simultaneously. A Result Viewer is created by registering the class using annotation and extending the Autopsy class *AbstractDataResultViewer* as shown in **Figure 9** for the class CamillaGraphBuilder.

```
@ServiceProvider(service = DataResultViewer.class)
@SuppressWarnings("PMD.SingularField") // UI widgets cause lots of false positives
public class CamillaGraphBuilder extends AbstractDataResultViewer {
```

Figure 9 Register a class for a Result Viewer module

Registering this class adds a new tab to the Listing tab, as shown in **Figure 10**.

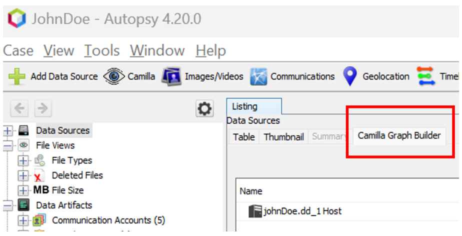

Figure 10 Camilla Graph Builder is created.

It is necessary to explore the relationships diagram among the Autopsy, NetBeans and base Java classes to understand how the registration of a class through the extension of *AbstractDataResultViewer* allows the rendering of a UI component. Figure 11 shows the relationships between the most relevant classes for the Plugin development, particularly that an AbstractDataResultViewer class extends a JPanel.

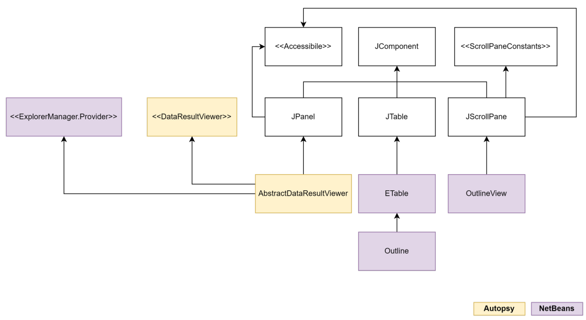

Figure 11 Diagram of the relationships between NetBeans, Autopsy, and base Java classes

Because of this inheritance relationship, it is possible to manipulate the UI using the Swing GUI framework, and integrating with any library of the vast Java ecosystem is straightforward.

5.3 Plugin Architecture

CAMILLA's foundational capability for graph modelling and visualisation leverages JGraphX, an Open Source library designed for the manipulation of graph nodes as visual elements within a panel. As depicted in **Figure 12**, the finalised plugin displays within the 'Camilla Graph Builder' tab, nested under the 'Listing' outer tab. Subsequently, CAMILLA divides the allocated user interface space into two distinct panels: the 'Outline' and the 'Canvas'.

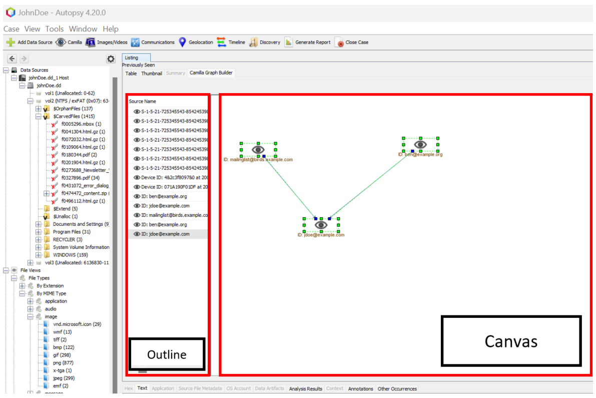

Figure 12 Main GUI elements of CAMILLA

The 'Outline' is a component derived from NetBeans, previously integrated within the Autopsy Core and subsequently extended by CAMILLA. Positioned on the left, this component actively responds to UI events originating from the left taxonomy, thereby displaying elements relevant to the selection. Intrinsically, the 'Outline' boasts a dynamic tabular structure, with its columns dynamically adjusting based on the nature of the artefact being investigated. Conversely, the 'Canvas' serves as a panel adept at visualising vertices and edges, achieving this through the library JGraphX. CAMILLA integrates these components — the Autopsy taxonomy, the Outline, and the Canvas — to create the interaction that empowers users to craft detailed graph diagrams of the entities' relationships. This integration permits elements from the 'Outline' to be seamlessly dragged onto the 'Canvas' and rearranged as deemed necessary. **Figure 13** delineates the interrelation between the various libraries in use and the foundational Java classes they extend. Notably, the classes emphasised in red suggest potential avenues for expanding the existing plugin to encompass a 'Timeline Builder', a feature not covered by this implementation but on which this thesis elaborates in the elaborated as part of the future work in the last chapter.

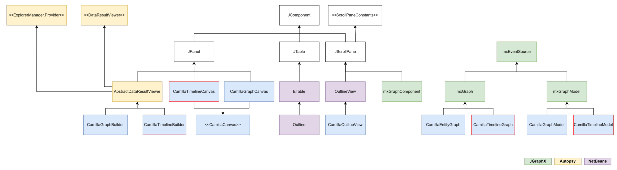

Figure 13 Class Diagram for CAMILLA

5.4 Camilla Graph Builder

The Graph Builder is realised through the class *CamillaGraphBuilder.java,* which, as discussed in 4.1, registers the component by extending the *AbstractDataResultViewer* class and splitting it into two parts (the Outline and the Canvas) using a JSplitPane.

5.4.1 Outline

The Outline is the UI component that reacts to the selection in the Autopsy taxonomy to show the artefacts of a given type. In CAMILLA, this is implemented through the *CamillaOutlineView* class. This class is quite complex and has been implemented as a fork of the base component available in NetBeans and used by the Autopsy Core module. The customised class extends the base behaviour, enabling Drag and Drop by leveraging the *Transferable* pattern. These changes enabled the dragging from the Outline into the Canvas, facilitating the creation of the visualisation.

5.4.2 Canvas

The Canvas is implemented through the *CamillaGraphCanvas.java* class, which extends JPanel. This class is responsible for everything that happens when the drop action is triggered. This class is reactive to the drag-and-drop events by adding a customised implementation of *TransferHandler, as shown in* **Figure 14**.

```
private class CamillaTransferHandler extends TransferHandler {

    @Override
    public boolean canImport(TransferHandler.TransferSupport support) {
        // Only accept the drop if the data being transferred is a Node
        // or if the dragged component is the graph component
        return (support.isDataFlavorSupported(new DataFlavor( type: Node.clas
    }

    @Override
    public boolean importData(TransferHandler.TransferSupport support) {
```

Figure 14 Implementation of TransferHandler to manage the drag and drop operations

The *canImport* method in the custom *TransferHandler* is responsible for assessing whether the data being dropped are of the expected type (a NetBeans Node, in this case). The *importData* function instead is responsible for manipulating the Node received from the Outline, unpacking its information, and publishing it on the graph in the form of a label to be shown below the vertexes icon, as depicted in **Figure 13**.

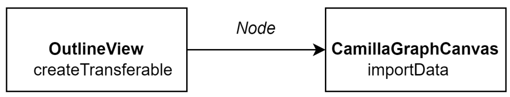

Figure 15 Objects transfer path

5.5 Utils

A utility class has been created to provide functionalities that don't naturally sit with the graph-building functionality. This class has been called **CamillaUtils.java** and implements several static methods to improve reusability should the plugin's scope be extended beyond Graph Visualization.

5.5.1 Save Visualisation

The ***saveVisualization*** static method accepts as an input an instance of a mxGraph, which is a JGraphX graph type, and saves it into the embedded SQLite instance of the current case. The method works by serialising to XML the input mxGraph instance using the capability offered by the ***mxXmlUtils*** class of JGraphX. The XML version of the graph is then saved in the SQLite instance of the case using an INSERT statement.
The serialisation has the adverse side effect of destroying the reference to the actual Autopsy artefact and is able to save only the name of the object rendered on the canvas for the time being.
This method has been designed in a way to accept the type enumerator parameter that specifies the type of visualisation being saved. This allows the same function to be able to handle multiple types as input, should the plugin be extended to support new forms of visualisations.

5.5.2 Load Visualisation

The **loadVisualization** method restores a graph on the canvas starting from the serialised version in the embedded SQLite instance. This method also accepts a VisualizationType enum as a mechanism to ensure future scalability when implementing other forms of visualisation beyond the entity graph. This method is invoked every time a case is opened for seamless integration between the plugin and the existing UX offered by Autopsy.

5.5.3 Save Graph to PNG

The main objective of this plugin is to facilitate the creation of rich reports. For this purpose, the ***saveGraphToPNG*** method can transform the image visualised on the canvas into a PNG that can then be downloaded and used as part of the DF report. A nuance of this implementation, a 50px padding, is created to improve the quality of the image so that the nodes on the canvas are not at the border of the saved image, as shown in ***Figure 16***, with the objective of providing something that can be added to a DF report without any other off-system manipulation in the interest of integrity and productivity for the end-user.

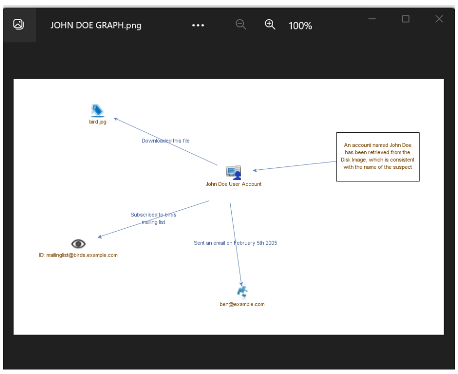

Figure 16 An exported visualisation from the John Doe case

6. RESULTS

This thesis has achieved the objective of creating a fully integrated plugin for Autopsy that allows DF professionals to manipulate found artefacts to create visualisations. The two pictures below show an example from the DonJoe case: here is a real comparison between what was created using an external tool (PowerPoint, **Figure 17**) and what has been possible to replicate using CAMILLA (**Figure 18**).

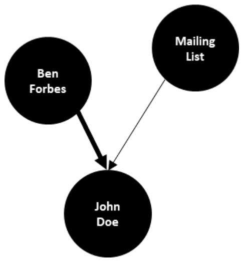

Figure 17 Relationship Diagram from John Doe Case

Figure 18 Entity Graph using CAMILLA (from John Doe Case)

There are multiple benefits to using the plugin. The most obvious, due to its convenience, is the fact that a separate external tool is no longer required. Another important element is that it is now possible to plot exact artefacts on the canvas that can be identified and referenced with the very same properties that the forensic tool (Autopsy) makes available. This means that it is now possible to create visualisations that have a 1:1 relationship with discovered artefacts, which is important because it allows to create visualisation of the relationships using the artefacts themselves, enabling traceability and minimise the chances of errors.

7 FUTURE WORK

Autopsy is a robust open-source platform currently undergoing active development. As of 29th August 2023, the latest version is 4.21. Given its capabilities, it is worth investing in to extend its functionalities. While this thesis successfully met its objectives, it merely scratched the surface of what's possible. By leveraging Autopsy, one can develop powerful tools that can potentially enhance the digital forensic reporting process, surpassing even what commercial software offers. Subsequent paragraphs will delve into potential future work aimed at further refining Autopsy to better serve the digital forensics professional community.

7.1 Improve User Experience

CAMILLA is released as fully working software that can be used in a production setting. The development has favoured the quality and stability of implementation over the quantity of features to achieve this objective. Therefore, whilst some improvements have been investigated, they have not made it to the release. One of the improvements that would make the UX even smoother is the full integration of the elements on the Canvas with what is displayed in the ContentViewer tab (**Figure 19**).

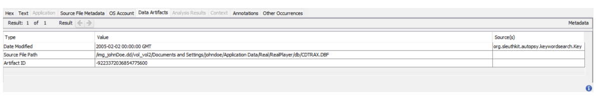

Figure 19 The ContentViewer tab

The vertexes displayed on the UI are active links to the Autopsy artefacts. However, a selection on the Canvas doesn't translate into the rendering of the contextual properties in the ContentView part of Autopsy. This can be considered a minor annoyance, given that the same information can be accessed by performing a right-click on the vertex (**Figure 20**). However, having the ContentViewer react to the selection on the canvas can enable further use cases like a dynamic exploratory data analysis: rather than knowing upfront which objects should be dragged on the canvas, multiple objects at the same time could be dragged in bulk, and at later stage visually investigated and linked.

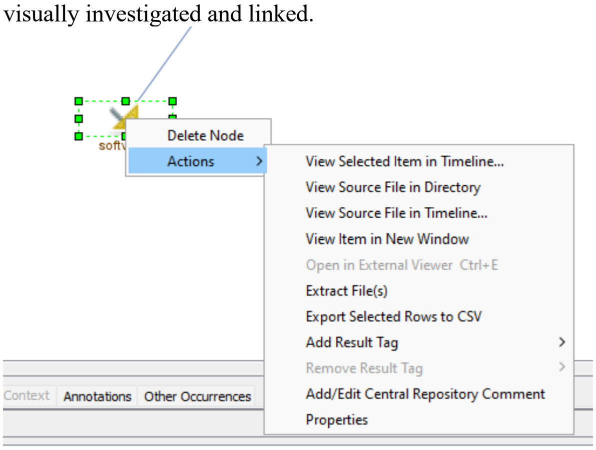

Figure 20 Contextual Menu on the canvas artefacts

Another feature that would benefit from another feature not yet implemented, which is the drag-and-drop of multiple objects simultaneously on the Canvas.

7.2 Improve Graph Visualisation

The current Graph Visualization functionality has room for improvement. Specifically, in the present version, saving the graph results in its serialisation in XML, which is then stored using SQLite. This process causes a disconnection from the original Autopsy artefact, making it impossible to access the artefact's properties via the contextual menu. A potential solution is to retrieve the Unique ID that Autopsy generates for each artefact and incorporate it into the serialised XML. When loading from this saved XML, the software should reference this unique ID to the artefacts stored in the case and reintegrate the reference into the graph's vertex.

7.3 Create a Timeline Visualisation

The role of DF experts goes beyond merely presenting uncorrelated facts or evidence in court cases, and they also provide expert judgment on events based on their uncovered evidence. One powerful tool that can enhance DF reports is the documentation of event sequences using timelines. While CAMILLA can currently plot a graphical representation of artefacts and maintain links to them, a valuable enhancement would be the introduction of a drawing modality. This would allow the plotting of selected artefacts on a timeline and the ability to annotate them. Autopsy already has a timeline functionality, but it is not practical for reporting purposes. A more effective representation of timing relationships can be seen in **Figure 21.**

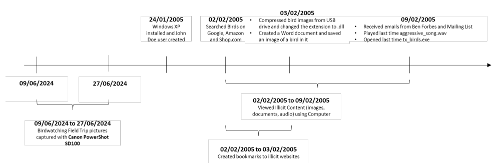

Figure 21 Timeline visualisation from John Doe case

7.4 Use actively developed graph library

Camilla employs a Java library named JGraphX to represent, visualise, and save entity relationships as graphs. However, as of 13th November 2020, this library is no longer actively developed, with its GitHub repository being archived. Utilising a library that lacks ongoing support is likely not a sustainable long-term solution. At the time this thesis was written, JGraphX seemed to possess all the necessary attributes to validate the thesis's objectives; thus, it was considered suitable. However, for long-term or commercial applications, it would be prudent to rely on contemporary software that, at a minimum, receives regular security updates.

7.5 Implement using web technologies

Autopsy, and by extension CAMILLA, is constructed as a desktop application utilising JavaFX and Swing. Although both these frameworks are mature and under active development, the prevailing trend leans towards employing web technologies for User Interface (UI) development. Consequently, any future development should evaluate the sustainability of continuing with a desktop-centric approach. Innovation in web technologies tends to progress more rapidly, and web development expertise is more abundantly available in the market. It might, therefore, be more advantageous to create a Java plugin that exposes necessary APIs through web services and delivers them using a UI crafted with one of the numerous web development frameworks that harness JavaScript.

7.6 AI DF Investigation and Reporting

At the time of writing this thesis, Large Language Models (LLMs) have obtained significant traction and are transforming the way people interact and leverage machines to achieve increasingly sophisticated tasks. These models challenge prior notions of what is achievable for machines and are being applied with increasing success across almost all domains and industries. With the ascent of LLMs, exemplified by OpenAI's GPT series and similar models, there has been a marked shift in the perception of artificial intelligence's potential. This thesis concludes with a Proof-of-Concept (POC) of what can be achieved by leveraging LLMs to enhance the Digital Forensics investigation and reporting process.

LLMs are excellent at processing, understanding and transforming Natural Language but have limitations – by design – when it comes to providing accurate and definitive answers. They are extremely versatile but cannot be applied blanket to any problem without considering those limitations that derive from their very nature. It seems, however, that they can be a valid and reliable help in supporting the DF investigative and reporting process.

7.6.1 CAMILLA as an enabler

Using CAMILLA, a DF professional can build a graph representation of everything that is retrieved using Autopsy and their relationships. Graphs help building a more intuitive understanding of the nature of relationships and support a rich representation of knowledge. By using CAMILLA it is possible to encode the relationships discovered during the investigative process, and save them in XML format. Since LLM can process language, not only Natural Language like plain English, this XML encoding can then be passed as an input. LLMs can process it and help the DF professionals in two ways: explaining in natural language the content of a graph, and recommending actions to be taken to further corroborate or refute a thesis. Used in this way, CAMILLA overturns the reporting process: a DF professional no longer needs to spend the time writing in Natural Language their understanding of what occurred in the report and then plotting the entity relationship in a way that documents their thesis but can investigate and make a hypothesis visually using CAMILLA and ask the LLM to decode the graph and write the report. This would free time from the DF expert that could be used in searching for evidence and correlating it and less in the mere task of documenting their understanding.

7.6.1 A POC using ChatGPT 4

This POC aims to demonstrate the feasibility of using LLMs to generate DF reports starting from graph structures created using CAMILLA. For this purpose, the simple diagram shown in **Figure 22** has been created based on the artefacts found in the John Doe case.

This POC uses ChatGPT-4 as the LLM of choice as it is universally considered the benchmark for generic-purpose tasks.

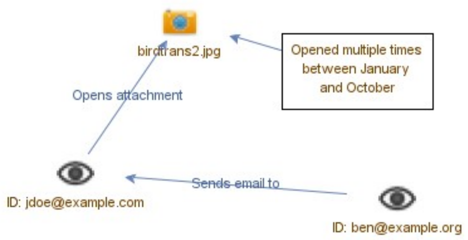

Figure 22 Simple example of entity-relationship graph

CAMILLA can represent the graph using XML before storing it in the SQLite database, which, in this case, is serialised as follows:

<mxGraphModel><root><mxCell id="0"/><mxCell id="1" parent="0"/><mxCell id="-9223372036854756864" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/ID:%20jdoe@example.com6632573924966326302.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="ID: jdoe@example.com" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="236.0" y="166.0"/></mxCell><mxCell id="-9223372036854756865" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/ID:%20ben@example.org3045273583970997127.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="ID: ben@example.org" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="492.0" y="186.0"/></mxCell><mxCell edge="1" id="29" parent="1" source="-9223372036854756865" style="" target="-9223372036854756864" value="Sends email to"><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="530.0" y="200.0"/><mxPoint as="targetPoint" x="290.0" y="190.0"/></mxGeometry></mxCell><mxCell id="-9223372036854775578" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/birdtrans2.jpg2446611888280790061.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="birdtrans2.jpg" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="319.0" y="46.0"/></mxCell><mxCell edge="1" id="30" parent="1" source="-9223372036854756864" style="" target="-9223372036854775578" value="Opens attachment"><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="280.0" y="180.0"/><mxPoint as="targetPoint" x="340.0" y="80.0"/></mxGeometry></mxCell><mxCell id="31" parent="1" style="shape=rectangle;strokeColor=black;fillColor=white;" value="Opened multiple times
between January
 and October" vertex="1"><mxGeometry as="geometry" height="60.0" width="120.0" x="440.0" y="70.0"/></mxCell><mxCell edge="1" id="32" parent="1" source="31" style="" target="-9223372036854775578" value=""><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="500.0" y="100.0"/><mxPoint as="targetPoint" x="340.0" y="50.0"/></mxGeometry></mxCell></root></mxGraphModel>

The above XML has been provided to ChatGPT using the below prompt:

This XML rapresent a graph of relationships between entities of a Digital Forensic case. In this scenario, all content that relates to birds is deemed illegal. Parse the content of the XML and write a Digital Forensics report of what you think is happening, what sort of relationships are in place and close with recommendations:
<mxGraphModel><root><mxCell id="0"/><mxCell id="1" parent="0"/><mxCell id="-9223372036854756864" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/ID:%20jdoe@example.com6632573924966326302.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="ID: jdoe@example.com" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="236.0" y="166.0"/></mxCell><mxCell id="-9223372036854756865" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/ID:%20ben@example.org3045273583970997127.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="ID: ben@example.org" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="492.0" y="186.0"/></mxCell><mxCell edge="1" id="29" parent="1" source="-9223372036854756865" style="" target="-9223372036854756864" value="Sends email to"><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="530.0" y="200.0"/><mxPoint as="targetPoint" x="290.0" y="190.0"/></mxGeometry></mxCell><mxCell id="-9223372036854775578" parent="1" style="shape=image;image=file:/C:/Users/danie/AppData/Local/Temp/birdtrans2.jpg2446611888280790061.png;verticalLabelPosition=bottom;verticalAlign=top;movable=1;" value="birdtrans2.jpg" vertex="1"><mxGeometry as="geometry" height="30.0" width="80.0" x="319.0" y="46.0"/></mxCell><mxCell edge="1" id="30" parent="1" source="-9223372036854756864" style="" target="-9223372036854775578" value="Opens attachment"><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="280.0" y="180.0"/><mxPoint as="targetPoint" x="340.0" y="80.0"/></mxGeometry></mxCell><mxCell id="31" parent="1" style="shape=rectangle;strokeColor=black;fillColor=white;" value="Opened multiple times
between January
 and October" vertex="1"><mxGeometry as="geometry" height="60.0" width="120.0" x="440.0" y="70.0"/></mxCell><mxCell edge="1" id="32" parent="1" source="31" style="" target="-9223372036854775578" value=""><mxGeometry as="geometry" relative="1"><mxPoint as="sourcePoint" x="500.0" y="100.0"/><mxPoint as="targetPoint" x="340.0" y="50.0"/></mxGeometry></mxCell></root></mxGraphModel>

ChatGPT-4 output is as follows:

Digital Forensics Report
Background:
The XML content provided represents a graph detailing relationships between entities in a Digital Forensic case. The content is visual in nature, with each entity represented as a node in the graph and relationships between them as edges or arrows. Based on the provided scenario, all content related to birds is considered illegal.

Findings:
Entities Involved:
jdoe@example.com: This seems to be an email ID that has been visualised in the graph.
ben@example.org: Another email ID that is represented.
birdtrans2.jpg: An image file, presumably containing content related to birds, given its name.

Relationships:
ben@example.org sends an email to **jdoe@example.com**: There's a direct relationship between the two email IDs, with **ben@example.org** being the sender and jdoe@example.com being the recipient.
jdoe@example.com opens the attachment birdtrans2.jpg: The relationship suggests that the image file birdtrans2.jpg was an attachment in the email sent by **ben@example.org** to jdoe@example.com.
birdtrans2.jpg was opened multiple times between January and October: This suggests that the recipient (jdoe@example.com) accessed the illegal content multiple times over a span of several months.

Analysis:
The graph indicates a digital communication between two users, where one user (ben@example.org) sent content related to birds (which is illegal in this scenario) to another user (jdoe@example.com).
The recipient (jdoe@example.com) not only accessed the illegal content but did so repeatedly over a considerable duration. This repeated access could indicate interest or involvement in the illegal content.
The attachment's name "birdtrans2.jpg" suggests that it might be one in a series ("trans2" could imply there was a "trans1" or more). This might mean that there are other related files, or this was a second transmission.

Recommendations:

Thorough Investigation: It's essential to conduct a more in-depth investigation into the email accounts of both jdoe@example.com and ben@example.org to determine the extent of their involvement with illegal bird-related content.
Check for Additional Files: The name of the attachment suggests that there might be more files related to this content. It would be wise to search for other related files on both jdoe@example.com and ben@example.org's devices and email accounts.
Evaluate Communication History: Examine the communication history between jdoe@example.com and ben@example.org for any other suspicious activity or exchanges of illegal content.
Legal Action: If it is confirmed that the content is illegal and both parties were knowingly sharing and accessing it, legal action may be necessary.
Educate and Monitor: Ensure that all users within the network are educated about the legal ramifications of sharing or accessing illegal content. It would also be beneficial to implement monitoring tools to detect and prevent the sharing of such content in the future.
Data Recovery: In case any data has been deleted or is hidden, employ data recovery tools and techniques to retrieve it and get a clear picture of the extent of the illegal activity.
Secure the Evidence: Ensure that all evidence, including the email exchanges, the image file, and any other related data, is securely stored and backed up. This will be crucial for any legal proceedings that may arise from the investigation.

As it is easy to observe, equipped with an extremely limited context on the nature of the case and the raw serialisation of the graph as an input, ChatGPT does a good job of extracting the semantics and putting the nature of the relationships it has derived in words. Moreover, it goes beyond the interpretation of the XML and suggests what actions should be taken to develop the investigation further.

These results are encouraging as the investigative process is creative in nature and does not have a unique answer, a circumstance that makes LLMs like the one implemented in ChatGPT a good candidate for an AI tool to support this process, and CAMILLA could be extended to generate – on the fly – the textual interpretation of what is being plotted on the CANVAS, making the entire process more controlled, predictable and, therefore, more helpful.

8. REFERENCES

Chappell, A.T. and Maggard, S.R. (2020) "Applying black's theory of law to juvenile justice decisions," *American Journal of Criminal Justice*, 46(3), pp. 461–482. Available at: https://doi.org/10.1007/s12103-020-09566-w.

Chen, H. *et al.* (2022) "A comparative study of automated legal text classification using random forests and deep learning," *Information Processing & Management*, 59(2), p. 102798. Available at: https://doi.org/10.1016/j.ipm.2021.102798.

Turner, M.J.L. (2021) *Computer evidence*, *Computer Evidence - Computer Misuse Act 1990 cases*. Available at: https://web.archive.org/web/20210911113635/www.computerevidence.co.uk/Cases/CMA.htm (Accessed: 25th March, 2023).

Karie, N.M. *et al.* (2019) "On the importance of standardising the process of generating digital forensic reports," *Forensic Science International: Reports*, 1, p. 100008. Available at: https://doi.org/10.1016/j.fsir.2019.100008.

Karstedt, S. (2021) "Inequality and punishment: A global paradox?," *Journal of Criminology*, 54(1), pp. 5–20. Available at: https://doi.org/10.1177/26338076211014590.

Ling, S., Kaplan, J. and Berryessa, CM (2022) "The importance of forensic evidence for decisions on criminal guilt ," *CrimRxiv* [Preprint]. Available at: https://doi.org/10.21428/cb6ab371.8c3889fd.

Olney, M. and Bonn, S. (2014) "An exploratory study of the legal and non-legal factors associated with exoneration for wrongful conviction," *Criminal Justice Policy Review*, 26(4), pp. 400–420. Available at: https://doi.org/10.1177/0887403414521461.

Peterson, J.L. *et al.* (2012) "Effect of forensic evidence on Criminal Justice Case Processing," *Journal of Forensic Sciences*, 58. Available at: https://doi.org/10.1111/1556-4029.12020.

Rattner, A., Turjeman, H. and Fishman, G. (2008) "Public versus private defense: Can money buy justice?," *Journal of Criminal Justice*, 36(1), pp. 43–49. Available at: https://doi.org/10.1016/j.jcrimjus.2007.12.005.

Sleuthkit (no date) *Releases · sleuthkit/autopsy*, *GitHub*. Available at: https://github.com/sleuthkit/autopsy/releases/ (Accessed: 03rd September 2023).

Jgraph (no date) Jgraph/JGRAPHX, GitHub. Available at: https://github.com/jgraph/jgraphx (Accessed: 03rd September 2023).

ChatGPT - Digital Forensics Relationship Analysis (no date) ChatGPT. Available at: https://chat.openai.com/share/949c3291-3fab-41d9-8c6e-89dc953c545a (Accessed: 04 September 2023).