European Commission | Horizon 2020 European Union funding for Research & Innovation

© 2018 SPIRIT ALL RIGHTS RESERVED

Deliverable 9.8

Privacy Controller for Modelling and Filtering Software and Report (b)

Planned delivery date: 31.07.2021
Submission date: 30.07.2021

Version 1.0

Work Package 9, Tasks 9.3, 9.4, 9.5

Deliverable lead partner: INN
Authors: Marco Tiemann (INN), Lesley Badii (INN), Ryan Faulkner (INN), Christian Weigel (FHG), Eva Blomqvist (LIU), Emma Teodoro Martinez (UAB)

Dissemination level	(√ where appropriate)
Public	√
Confidential, only for members of the consortium (including the Commission Services)	
Classified, as referred to Commission Decision 2001/844/EC	

Changelog

Version	Date	Changes	Authors
0.1	03.06.2021	Initial ToC release	Marco Tiemann (INN)
0.2	11.06.2021	First draft	Marco Tiemann (INN), Ryan Faulkner (INN), Lesley Badii (INN)
0.3	03.07.2021	Document updates	Marco Tiemann (INN), Lesley Badii (INN)
0.4	17.07.2021	Incorporation of ethics sandbox development activities	Marco Tiemann (INN), Emma Teodoro Martínez (UAB), Eva Blomqvist (LIU)
0.5	28.07.2021	Incorporation of video processing activities	Christian Weigel (FHG), Marco Tiemann (INN)
0.6	28.07.2021	Document review improvements	Lesley Badii (INN)
1.0	29.07.2021	Document release version	Marco Tiemann (INN)

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 786993.

Table of Contents

Horizon 2020
European Union funding
for Research & Innovation

Table of Figures

Table of Tables

Executive Summary

This deliverable provides the final documentation of the system designed and developed in the SPIRIT project for the purpose of ensuring that legal, ethical and privacy requirements concerning the SPIRIT platform can be addressed by the software components developed in WP9. To this end, the following components are covered:

- Authentication and authorisation
- Activity Logging
- Privacy Controller including analytics
- Analytics
- Anonymisation use cases and techniques

In this deliverable we have revised and updated the information provided in the SPIRIT deliverable D9.7 “Privacy Controller for Modelling and Filtering Software and Report (a)” in order to represent the final state of the work presented. While the majority of content in the deliverable have been revised and updated, some parts and descriptions, specifically Section 3 of this document, remain largely unchanged as they continue to correctly represent the work carried out. Sections on analytics and anonymisation methods extend the scope of this deliverable relative to the superseded deliverable D9.7.

1 Introduction

This introductory section covers the background of this deliverable, its purpose, the relation to other work in WP9 and explains the organisation of the remainder of this document.

1.1 Background

The SPIRIT project includes a dedicated work package WP9 concerned with ethical, legal and privacy aspects of the project and the SPIRIT Platform. In WP9, two main activity streams are distinguished, of which one identifies and addresses legal, ethical and privacy elements concerning the project in a broad sense which includes ensuring the compliance of the project as well as of the output of the project with relevant legislation and various external and internal requirements and guidelines. The second activity stream in WP9 has created relevant system infrastructure to address legal, ethical and privacy challenges as part of the SPIRIT Platform. While the boundaries between the goals and activities of these two streams have remained soft and collaboration among the two strands has been very important throughout the project, the work reported on in this deliverable is concerned with the software designed and implemented to address the relevant challenge areas and has been written as a technical document to accompany the software contributions created by Work Package 9.

1.2 Purpose

The purpose of this document is to technically describe and document the software components designed and developed as part of the work in WP9. The intended target audiences of this description are:

- Developers and partners within the project consortium as well as ones involved in post-project development activities who will find a comprehensive source of documentation that is intended to help them in their work on the SPIRIT Platform
- Technical domain experts who are interested in the system design and implementation and would like to understand these at a technical level (e.g. in order to gain insights into how such systems can be constructed and which factors may need to be taken into consideration when doing so)
- Potential users of the SPIRIT Platform and persons involved in assessing ethical and privacy compliance of software such as the SPIRIT Platform who need to understand how functionalities concerning legal, ethical and privacy have been designed and implemented in the SPIRIT Platform (e.g. in order to understand the level of compliance supported by the SPIRIT Platform)
- Interested members of the public who may for instance want to learn how EC-co-funded projects safeguard their rights in sensitive application domains

In addition to technical descriptions, this document also includes guidance and how-to documentation as appendices to the document main body. Not all the appendices of the deliverable are included in the final public release of the deliverable; some of the deliverables contain confidential configuration information and are hence omitted from the public deliverable version. The confidential release of this deliverable may be made available on request.

1.3 Context Within WP9

This document is the second and final deliverable that specifically addresses the technical implementation work in WP9 – the bulk of the other deliverables in the work package are addressing legal, ethical and privacy considerations from a broader perspective and do not directly address implementation details of systems specifically developed in order to safeguard users, deploying organisations and data subjects of the SPIRIT platform. It is useful to clarify the integration between the technical development and the broader activities within WP9 for clarity.

The technical developments in WP9 implement systems that enable robust logging and monitoring of activities carried out using the SPIRIT platform; they implement all the basic system required for this purpose. The collaboration with the broader tasks of the work package in this area of work ensure that all system activities that should be logged are logged as required for analyses, that activities that are not carried out directly through the platform but are relevant for compliance are identified and that appropriate measures are taken to address them at the user interface level (e.g. by requiring explicit confirmations that relevant authorisations that are granted outside of the scope of the SPIRIT platform have been granted), that the gathering of such log data itself complies with applicable legal requirements including ethical, legal and privacy regulations as applicable.

Based on the developed infrastructure for logging and monitoring, an analytics and intervention layer is part of the software developed in WP9. Here, the integration with the general requirements for legal, ethical and privacy protection are directly interpreted and integrated into the system decision making processes so that the technical systems implemented support the overall goals and requirements of the work package and the project as a whole in terms of legal, ethical and privacy protection.

In addition to these activities, additional efforts have been undertaken in order to enable the integration of ontology models that aim to formally represent the legal environment and requirements in which the SPIRIT Platform operates. This includes reviews of existing high-level legal ontologies in terms of their applicability as well as in terms of the extent to which they can be used in a system such as the SPIRIT Platform.

As for the overall approach in the work package, some of the requirements and implementations differ due to the legal environments in different participating countries, so that both the overall WP9 work and the technical work are also concerned with ensuring that these country-specific (and potentially also organisation-specific) requirements can be represented through custom configurations in the software implementation.

From a process view, requirements specification and system implementation for legal, ethical and privacy protection have formed a cycle of frequent iterations where those partners concerned with the overall view on these issues provide feedback and input for the software implementation in regular intervals. This process was intensified in the final phases of the project in connection with the SPIRIT ethics sandbox activities.

1.4 Organisation

The remainder of this document is organised as follows: Section 2 introduces the overall subsystem structure and briefly introduces the components of the subsystem as a whole. The section also describes the integration of the subsystem into the overall SPIRIT Platform. Section 3 describes the user authentication and authorisation component implemented for SPIRIT. Sections 4, 5 and 6 describe the components that are developed specifically for the project: activity logging, user interface components, the Privacy Controller component and its analytics functionalities. Section 7 describes use cases and approaches for output processing, particularly concerning anonymisation use cases and techniques. Section 8 concludes the deliverable main body with summary remarks and an outlook towards developments beyond the end of the project. Two appendices provide more detailed technical reference information primarily targeted at the relevant partners within the SPIRIT consortium; as noted above, not all of the appendices produced are included in the public release of this document.

2 The SPIRIT Legal, Ethics and Privacy Subsystem

This section gives an overview over the subsystem covered in this deliverable as well as its integration with the rest of the SPIRIT Platform.

The SPIRIT Privacy, Ethics and Legal (or PEL) subsystem is a set of components that are integrated into the SPIRIT Platform. To explain the terminology used to distinguish the parts covered in this deliverable: the "subsystem" is the overall set of parts that address legal, ethical and privacy concerns and requirements in the platform; "components" are distinct elements or groups of elements with internally similar functional focus that implement some functionalities used as part of the overall subsystem.

2.1 Components

The components making up the PEL subsystem can be grouped as follows: a) authentication and authorisation, b) activity logging, c) analytics, d) output processing and e) user interaction. Each of these five is described in the subsections below.

2.1.1 Authentication and Authorisation

Authentication and authorisation functionalities are important basic functionalities required in order to enforce legal, ethical and privacy protection as part of the SPIRIT platform – the system must be capable of restricting access to system functionalities and/or data in order to meet legal requirements and expectations by end user partners.

Authentication and authorisation functionalities are provided by a customised Apache Syncope[1] service. Apache Syncope is a system for identity management which provides functionalities for user account and rights management of external resources.

While Apache Syncope implements functionalities to define and retrieve authorisation information, authorisation rights and restrictions need to be enforced throughout the SPIRIT platform. This has been achieved through close collaboration with the partners responsible for the system front end and user interface and with the lead partners that implement the system backend orchestration. More information concerning this can be found in the relevant deliverables focusing on the user interface and backend development.

2.1.2 Activity Logging

Activity logging is the key functionality of the subsystem that is both necessary as a functionality in and of itself that enables system activity audits that may be required legally or as part of organisational policies, and in order to enable the other functionalities of the PEL subsystem.

We have to reconcile two technical requirements when implementing activity logging: the overriding requirement for the SPIRIT platform is to gather all relevant activity logging data that is required for the various

[1] https://syncope.apache.org/

system functionalities; the second requirement is to minimise the implementation effort in particular faced by the technical partners not involved in WP9 in order to reduce the potential for errors and unnecessary overhead when implementing activity logging.

We address this by attempting to gather all relevant backend data through the RabbitMQ messaging system that is the central communications hub for the SPIRIT platform backend. For activity logging that cannot be gathered through the SPIRIT RabbitMQ messaging system, two additional mechanisms have been added: a REST API to be used by partners in order to submit activity logging data and read access to data and in particular processing metadata that is entered into the SPIRIT database by processing components. The REST API in particular is used by the user interface development team, necessitating close collaboration in order to identify user interface functionalities that require logging calls.

2.1.3 Analytics

Data gathered through activity logging needs to be analysed in order to identify actions to be taken by the system and markup of activity logging data in the user interface. The Privacy Controller component is used to analyse activity log data as they arrive; it also retrieves data from the SPIRIT database for analysis purposes.

The analysis in SPIRIT is primarily carried out using rule-based reasoning mechanisms, so that relevant rules and requirements can be defined explicitly and can be modified to create custom sets for actions to trigger depending on individual end user partner requirements. An ontological reasoning subsystem has been integrated in order to facilitate the integration of reasoning mechanisms into the rule-based processing component. In addition, an open source stream mining tool has been integrated as a technical demonstration of how statistical machine learning techniques can be integrated into the subsystem.

2.1.4 Output Processing

One expected output coming from the analytics functionalities of the PEL subsystem is to be able to automatically modify processing results of the SPIRIT platform in order to ensure legal, ethical and privacy compliance. This can include functionalities such as masking faces of irrelevant persons in video footage or similar activities as well as simpler functionalities such as dispatching notification messages that a pattern of user behaviour that should be reviewed has been observed.

As there were no practically relevant use cases identified for this component that could be evaluated within the scope of the project and available data sources, output processing techniques and their integration are discussed in Section 7 of this document but have not been implemented in the final demonstrator.

2.1.5 User Interaction

Activity log data as well as analytics results need to be accessible to (appropriately authorised) system users so that these users can review and scrutinise the activities of users and of system components when necessary. Hence, the PEL subsystem includes a user interface component that lets users access activity log data and associated analytics results. This functionality is provided by a dedicated user interface for the Privacy Controller, which reflects the separation of concerns in terms of usage of the system and enables more effective exploitation of the subsystem by partner Innova Integra.

European Commission | Horizon 2020 European Union funding for Research & Innovation

2.2 Integration

The PEL subsystem is integrated into the overall SPIRIT and interacts with a number of other subsystems and components of the platform. The following figure illustrates the interactions; it is explained further below.

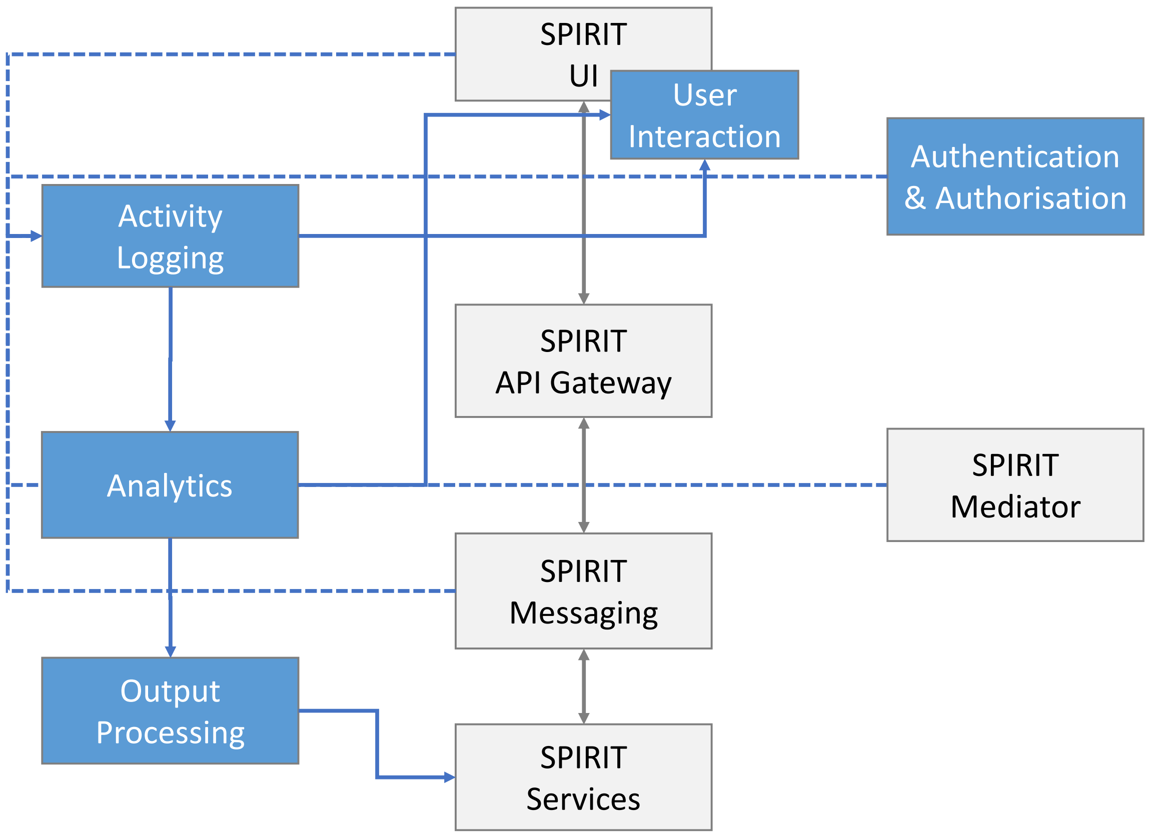


Figure 1: PEL Subsystem Integration in the SPIRIT Platform

Grey boxes indicate non-PEL platform subsystems and blue boxes indicate PEL subsystem components as introduced earlier. Grey arrows indicate interactions between the non-PEL platform subsystems; these are not of concern here. Dashed blue lines indicate log data flow. Full blue lines show output data flows from PEL subsystem components.

Figure 1 shows the component integration diagram as identified at the time of writing this deliverable. It is expected to evolve over time as specific additional technical needs emerge.

As indicated earlier, several SPIRIT subsystems are data sources for activity logging data:

- The SPIRIT User Interface send activity logging data to the Activity Logging component via the RESTful logging interface;
- The Authentication & Authorisation component provides data on activities carried out that alter user accounts and user rights via the RESTful logging interface;

- The SPIRIT RabbitMQ messaging system provides copies of messages sent through the system directly to the Activity Logging component using a separate messaging queue for that component; the messages provided cover the backend activities of the SPIRIT system.
- The SPIRIT Mediator as the central database management system of the platform provides access to the analytics data in the system as well as to the analytics metadata that are stored alongside the analytics results data.

The Activity Logging component acts as a data provider to other components within the PEL subsystem. This includes the PEL User Interaction component, which receives input from the Activity Logging component as well as the Analytics component and is the access point for displaying gathered and analysed data to authorised users. The Output processing component is designed to interact with the relevant SPIRIT Services for analytics that are part of the SPIRIT backend processing workflow as appropriate.

While the preference for the integration of the PEL subsystem as part of the overall SPIRIT platform is to require as few as possible custom integrations with other components, some such integrations are necessary; the integration with the user interface system is the most extensive and most noteworthy one in this respect.

3 User Authentication and Authorisation

User authentication and authorisation are key areas of functionalities that the platform needs in order to be able to meet requirements concerning privacy, data protection, legal requirements and auditability. Hence, this group of functionalities is provided as part of WP9 technical activities as it is the best fit within the project WP structure. This section characterises the selected implementation and the functionalities provided as part of the SPIRIT platform. Appendix A provides detailed technical documentation and usage instructions for project partners involved in interacting with the authentication and authorisation component directly. Please note that this Appendix is not included as part of the official public release of the document.

As mentioned in the introduction to this document, this section remains unchanged relative to the description in the previous deliverable iteration D9.7. Persons who are familiar with that deliverable may skip this section. The section has been retained for the benefit of readers who have not previously used deliverable D9.7. Appendix A has been modified to reflect minor updates to defined user roles, groups and demonstration user accounts. All data provided in Appendix A in D9.7 remains accurate; however it is recommended to refer to the updated Appendix presented in this deliverable for the most recent release of the documentation.

3.1 Implementation and Deployment

Authentication and authorisation functionalities are standard functionalities that are necessary in many modern software applications, and the requirements specified for the SPIRIT platform necessitate user authentication and authorisation mechanisms for different purposes and reasons. It was hence decided to use an off-the-shelf application to manage authentication and authorisation functionalities in order to speed up development, and in order to be able to base developments on an existing application with an established installation base and ongoing development efforts.

Apache Syncope[2] was selected as a suitable platform based on a review of the features required for SPIRIT and taking into consideration prior familiarity of the relevant developers with the system. While Apache Syncope is a system originally designed in order to integrate heterogeneous user identity management systems through a single middleware, its functionalities provide a complete user identity management system and enable fine-grained definitions of user authentication and authorisation functionalities. Apache Syncope is available as open source software, so that custom functionalities can be developed using the platform source code (provided relevant license requirements are adhered to). The availability of the system source code was necessary in order to be able to add activity logging calls to the external activity logging REST endpoint for SPIRIT. In technical terms, a custom Docker image was created extending the default Apache Syncope Docker image provided by the development team in order to carry out the system adaptations necessary for the project.

Apache Syncope does not support GraphQL, and it was furthermore considered desirable to store data concerning user authentication and authorisation in a dedicated database that could be secured restrictively

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

given the expected system configuration. Hence, a PostgreSQL database Docker image has been adapted from the publicly available Docker image released by the system developers. Upon first deployment on a new host machine, the databases required for Apache Syncope are automatically created and initialised with a custom system configuration for use in SPIRIT.

Since the SPIRIT platform is deployed on dedicated local machines provided to end user partners during the project, each such installation is serviced by a dedicated local Apache Syncope installation. In later commercial deployments this may be modified; furthermore, the abilities of Apache Syncope to integrate with existing identity management systems may be exploited in order to integrate a commercial SPIRIT system more easily into an existing IT infrastructure (this can also be extended to replace username/password authentication with different mechanisms).

3.2 Functionalities

This subsection briefly characterises the system functionalities provided by the authentication and authorisation component. All of the described functionalities are provided through a REST API; other interfaces that are provided by Apache Syncope have been disabled for safety reasons but may be re-enabled if users in the project need to use them.

The following table outlines the functionalities provided by the customised Apache Syncope implementation. Details on individual REST web service methods are given in Appendix A and in the online documentation for the Apache Syncope REST interface available after deployment of the two necessary Docker images.

Table 1: Authentication and Authorisation Functionalities for SPIRIT

Functionality	***Description***
Authentication	
User account CRUD features	The component provides mechanisms for creating, reading, updating and deleting user accounts for use in SPIRIT.
User account property management	The component defines a number of user properties which can be assigned to users, including associations with police forces.
User authentication and authentication token management	The component provides a means to authenticate users (using username/password combinations for the development system) as well as to manage, verify and expire authentication tokens that can be used to validate authenticated users.
Authorisation	
Definition of user groups and roles	The component provides a mechanism to define user roles and groups that can be used to assign rights based on custom-defined user roles and group memberships.

Definition of arbitrary authorisation targets and relations	The component provides a simple graph mechanism which can be used to describe arbitrary authorisation relations.
Authorisation data CRUD features	The component provides mechanisms for creating, reading, updating and deleting authorisation data for use in SPIRIT
Logging	
Logging of all user and system activities	The component provides a built-in mechanism for logging all actions carried out in the system; the logging data provides a sufficient amount of log data detail for use in SPIRIT.

This summarises the functionalities provided to the SPIRIT project by the Apache Syncope installation.

3.3 Data Model

This subsection briefly summarises the data model used to represent users in SPIRIT. A full documentation of the user model and all individual properties of it is available to developers in the confidential Appendix A of this deliverable.

Table 2: User Data Model Property Types for SPIRIT

Data	***Description***
User account data	Unique user identifier, email address, password
User account data for interaction purposes	User first and last name
User affiliation data	Organisation membership
User role data	Assignment to system-functional user roles – these roles are defined as roles that group functionalities for use in the SPIRIT platform as opposed to organisational roles as default; role assignments can be modified in instances where organisational roles can be translated directly into system-functional user roles. User roles are used to assign all general (i.e. not investigation-specific) user rights concerning system activities. This includes for instance user role definitions for viewing general activity log data.
User-investigation rights	Assignment of various types of rights relating users to investigations; investigation data is not maintained within the Apache Syncope database and investigations are referenced in the system using a unique identifier only. Associations between a user

European Commission | Horizon 2020 European Union funding for Research & Innovation

	and an investigation include “has-read-access”, “has-write-access” and the like and can also include specific rights such as a “has-darkweb-search-authorisation”.

This table describes the groups of user data model properties of the final PEL subsystem prototype released in the project. The model is primarily used in order to facilitate user role management and the assignment and management of users to investigations in order to manage investigations.

More complex rights structures can easily be implemented using the available infrastructure but were not considered necessary within the scope of project development.

4 Activity Logging

Activity logging is a core activity of the SPIRIT PEL subsystem – it provides the vast majority of the data that is expected to be used by the Privacy controller component for data analyses and gathering and retaining activity log data is important in order to document user and system activities and in order to allow for scrutiny of the SPIRIT platform in action.

4.1 Data Model

The Activity Logging component expects a standard data model to be submitted through the Activity Logging REST API, and it transforms messages that it intercepts or receives via the RabbitMQ message queueing service into the standard data format. The format is also used in order to represent data that is retrieved from the SPIRIT Mediator.

The activity logging data model is defined as a JSON document that contains the following components (displayed here in table format to improve legibility over JSON text format). Elements in bold type face are mandatory for all activity logging elements.

Table 3: Data Model for Activity Logging "Activity" Element

Element	***Description***
id	Unique identifier for an activity log entry, will be assigned by the system and cannot be assigned by you when using a log endpoint
username	Username, should always be included in calls
bearertoken	Bearer token, should always be included in calls when it is available; can be omitted when it is not available (e.g. before authentication or in cases where a system component does not receive a bearer token)
isValidated	System property concerning user validation, any values submitted by users during creation will be reset to false, so should be omitted to save bandwidth
time	Time of the action in ISO8601 format as follows: "2019-07-29T15:26:26+00:00" with offset for a time zone used or "2019-07-29T15:26:26Z" when using UTC time; the time reported should be as close as possible to the time of the action
source	The source of the activity log entry. This is the component id of the system sending the information of concern in the entry.
target	The target of the activity log entry. This is the component id of the system receiving the information of concern in the entry. This may be omitted if there is no receiving system involved.

action	The action being reported as defined in the action table in this document
payload	A list of PayLoad elements that contain the message data that is being sent. The format of PayLoad messages is described in the table below. Each Activity may contain a list of PayLoad elements; where possible, the elements should be represented in order to occurrence. PayLoad elements are also used to represent data retrieved from the SPIRIT Mediator.
parameters	A hash map of parameter key/value pair String data that document component configuration parameters have used for processing. The parameters used should document all parameters settings that deviate from the default settings used for the component version that is used and referenced in the "source" element of this Activity element
output	A list of references to output generated by the system that is stored somewhere; can either be an internal path to a file or a URL referencing a specific request to the SPIRIT Platform database

The PayLoad elements referenced in the table above contain message payload data, which contains processing data and/or metadata depending on whether it contains data from REST API calls, RabbitMQ messaging or data retrieved from the SPIRIT Mediator. Table 4 describes the data model for the PayLoad[3] element.

Table 4: Data Model for Acivity Logging "PayLoad" Element

Element	*Description*
id	Unique identifier for a PayLoad entry, will be assigned by the system and cannot be assigned by you when using a log endpoint
path	The path that you are accessing with a call; for a REST call this would be the URL used for the call. If a call is made in a different format, please identify the nearest equivalent to a path (e.g. "database name:table" name for a database update)
mode	The mode in which you are making a call; for a REST call this would be the method used (e.g. GET or POST). If a call is made in a different format, please try to identify the nearest equivalent to a mode, unless that information is already contained in the path or body, then leave mode empty.

[3] This element was previously named "ApiCall". It has been renamed in order to correctly apply also to widened scope of usage in particular as it is also used to retrieve data from the SPIRIT Mediator.

body	The body of a message. For a REST call this would be the message body; for a RabbitMQ message this would be the content of the message sent to the relevant queue. In instances where copies of write submissions to the database are communicated, the "create" or "update" instruction should be included, but the values to be inserted do not need to be provided (while fields do). ***Please note:*** *all body information* ***must*** *be sent base64-encoded in order to ensure that valid JSON data is transmitted to the logger. The system will always base64-decode at the receiver end, so information sent as body must be base64 encoded in* ***all*** *cases.*

"Source" and "Target" definitions in "Activity" elements are required to be provided in a specific format that captures data about the specific system components and specific component versions involved in an activity logging entry. Appendix B provides a table elaborating on the related naming scheme requirement.

4.2 Logging API

The Activity Logging API is provided as a REST API to which components send activity log data through calls. Components submit data through a "log" web method call; the API provides methods for logging a single activity log entry and a list of activity log entries. The API exposes a query interface so that other components can retrieve log data for display and/or further processing. This API is only used by PEL subsystem components, namely the user interface component and the Privacy Controller.

4.3 Message Queueing System Message Interception

The SPIRIT platform backend uses RabbitMQ as a message broker for carrying out tasks in the analytics backend of the platform. This makes the RabbitMQ broker an excellent point for gathering activity log data, as the backend components communicate their status and activities through the message broker. This approach minimises the effort required by backend module developers and reduces the potential for human error, because these developers are not required to actively send copies of messages for logging purposes.

The technical implementation uses a feature of RabbitMQ originally intended for debugging purposes. RabbitMQ is configured to automatically send a duplicate copy of each message sent to any RabbitMQ queue to a second queue (together with additional metadata concerning the original message) that is used exclusively by the Activity Logging component. The Activity Logging component receives these messages through a custom RabbitMQ client. To facilitate this way of receiving messages as they are transmitted over RabbitMQ, a customised version of RabbitMQ has been implemented as part of the WP9 development work. Development of the custom version does not involve source code development. It is realised via specialised start-up and configuration scripts for RabbitMQ. These scripts are provided in Appendix B for the convenience of developers. Please note that the that Appendix B has been omitted from the public release version of the deliverable, as it provides specific configuration details that may be used in order to degrade system performance under specific circumstances.

4.4 SPIRIT Mediator Data & Metadata Retrieval

Some SPIRIT subsystems that carry out data retrieval and analysis activities store – in addition to processing output data that are primarily used as auxiliary data by the PEL subsystem – also specific metadata that relate to processing applied, time taken and other data of relevance to the PEL subsystem that are not part of the messaging routed but are relevant as logging input. In addition, the SPIRIT Mediator itself adds metadata relating to data storage to the database which is relevant in order to ascertain the timing of some data processing activities.

The SPIRIT PEL subsystem integrates a query mechanism that automatically schedules specific queries that retrieve data from the SPIRIT Mediator when a SPIRIT subsystem that stores additional data in the Mediator sends a relevant processing instruction via the RabbitMQ messaging system. The mechanism can be used to requeue data queries in order to retrieve data for incremental or long-running processing activities.

For convenience and in order to simplify the processing pipeline, data retrieved from the SPIRIT Mediator is converted into the common pipeline data format described in Section 4.1.

4.5 Message Transformation

All of the data sources above can supply different message formats to the Activity Logging component:

- In the case of REST API logging, the system expects largely uniform data where the payload content differs depending on the type of logging message sent.
- In the case the of RabbitMQ messages, the message format and content differ for the individual data analytics component except for a general framework format for the message queue system.
- In the case of the SPIRIT Mediator query data, the message format represents the retrieved data uniformly, but the data that is retrieved differs by component and associated query or queries executed, as well as by the execution result in cases where data are not (yet) available in the SPIRIT Mediator.

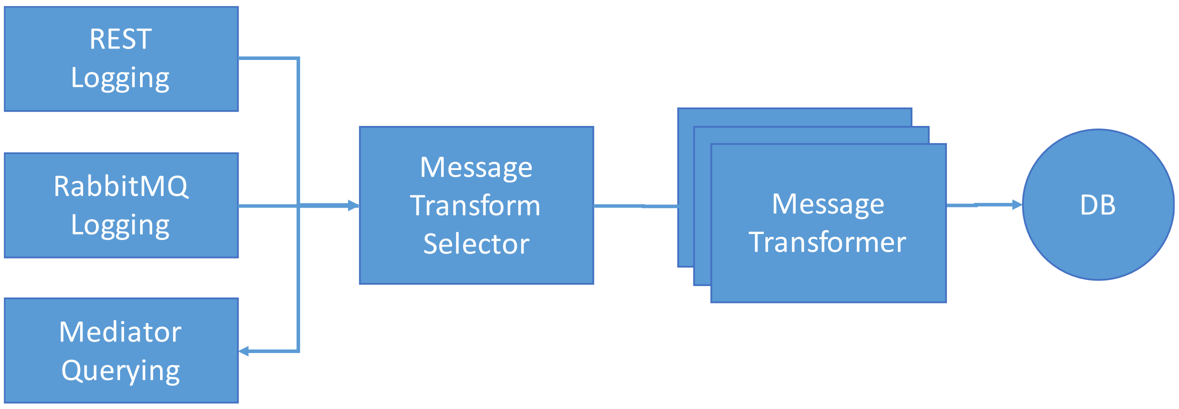


Figure 2: Message Flow Through Activity Logging Component

Horizon 2020
European Union funding
for Research & Innovation

In order to create suitably uniform activity logging data, all three data sources are piped through data transformers that convert the incoming messages into the final activity log entry that is stored in the activity logging database. A plug-in infrastructure is used in order to be able to maintain and add data transformer modules to the system as the messages sent through the system are expected to change throughout the course of the project. Figure 2 shows a summary view how messages move through the Activity Logging component through to being stored in the database.

4.6 Integrity of Log Data Storage

The activity log data gathered by the Activity Logging component are the basis of further analyses and are the data that are presented to auditors when reviewing SPIRIT system activities. Because of this, it is of particular importance to protect the gathered log data, in particular from unauthorised alteration and deletion of log data entries by malicious external or internal adversaries.

As a first step towards this, the SPIRIT PEL subsystem stores log data in a separate database management system that is not easily accessible to end users or users with administrative access to the main SPIRIT Platform system.

As a second step, the SPIRIT PEL subsystem generates unique validation information for log data that is added to the logging database. Log data entries into the log database are not modified once they have been added to the logging database and so can be considered to be immutable. This allows us to use mechanisms based on which we can identify whether data has been modified in order to check for any tampering that may have occurred. The topic of detecting tampering attempts in log data has been considered in data security research including for database log data (see e.g. [SNO04]).

Generally, approaches involve generating hashes for log data entries that cannot easily be recreated by someone modifying log data and that can be analysed in order to identify when data in logs have been modified. Key trade-offs in this application concern the cost of generating hashes, where generated hashes are stored and how they relate to other entries in a log.

For the PEL subsystem log storage, we use cryptographic hashes where groups of log data entries are hashed together using a Public Key Infrastructure system of private encryption and public decryption keys. Hashes are generated using the data of the groups of log data entries being hashes as well as the hash data of the previous set of hashed log data entries as demonstrated graphically in Figure 3 (excluding special cases such as initialising new log databases).

European Commission | Horizon 2020 European Union funding for Research & Innovation

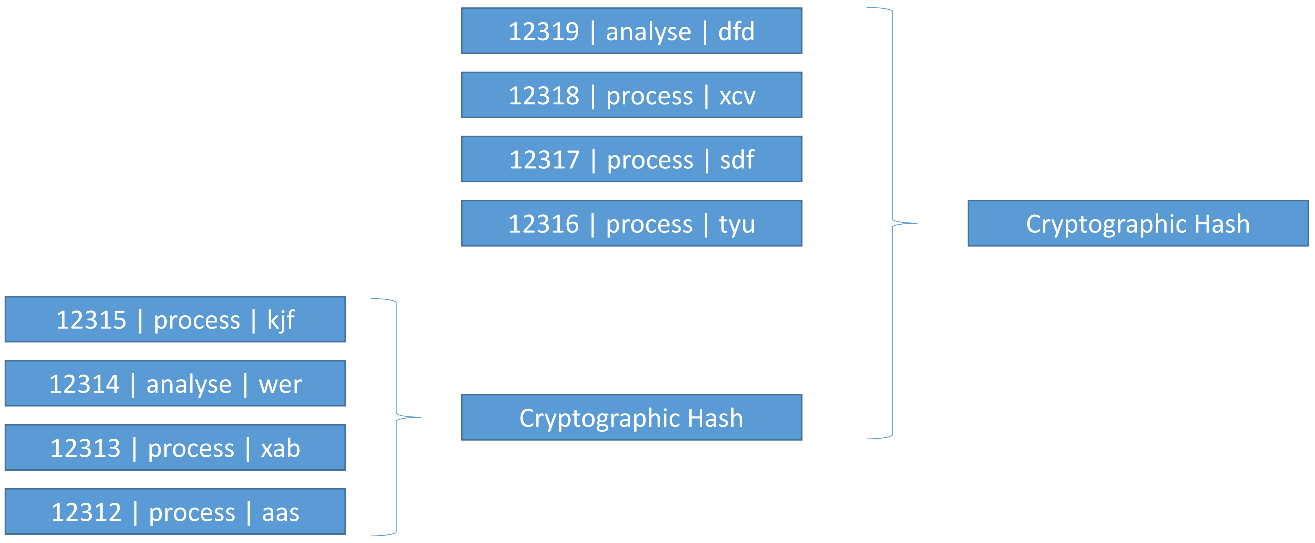


Figure 3: Graphical depiction of block-based hashing for log data

This encryption mechanism has some similarities with ledger-based approaches that are used for crypto currencies and have received a lot of attention in recent years. The implementation for the PEL subsystem is a proof-of-concept solution that uses a simple PKI library and hashing mechanism in order to demonstrate the concept.

4.7 Conclusion

This concludes the description of the data acquisition and storage of logging data gathered as part of the SPIRIT PEL subsystem. The next section describes the data processing over these data in the Privacy Controller.

5 Privacy Controller

The Privacy Controller is the data processing and analytics component of the SPIRIT PEL subsystem. This section outlines the composition of the component and gives an overview over the Privacy Controller system and the module implementations within the scope of the project.

5.1 Overview

The Privacy Controller component implements analytics modules (referred to as Analytics Processors) that analyse primarily activity log data and in addition also further data stored in the SPIRIT platform if needed in order to process activity log data. The Privacy Controller evaluates available log data in terms of identifying whether user and/or system activities require review by a human operator, because they may indicate an ethical, legal and/or privacy issue having occurred during processing.

Figure 4 depicts the processing workflow of the Privacy Controller. Activity logging data are pushed to the Privacy Controller Data Gatherer module, which initiates a workflow. The Process Manager module evaluates the message received and activates a relevant Analytics module if considered relevant. That module may retrieve additional data form SPIRIT data sources, including actual source data such as source texts or images from the SPIRIT platform file repository if necessary. It then analyses the data available and stores the generated output into a result database.

Where necessary, it can also trigger a Privacy Controller Actuator module that is integrated into the SPIRIT platform workflow so that it can modify output generated by other modules, primarily for privacy protection purposes (see Section 7).

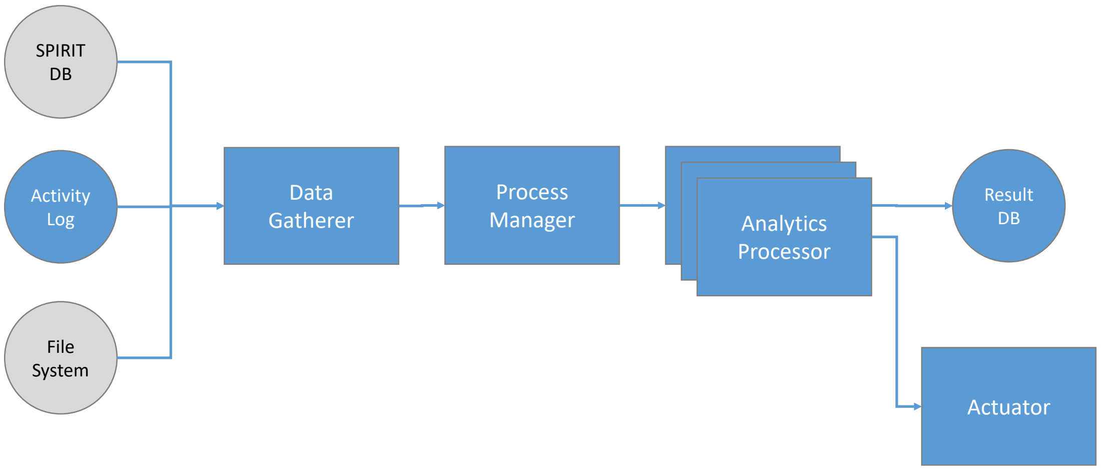


Figure 4: Overview of Privacy Controller Processing Chain

The Data Gatherer and Process Manager components depicted in the diagram are workflow processing components that parse and forward data to analytics components as deemed relevant and that manage user interactions via the user interface. The Result DB stores analytics results for display and notification to auditors.

European Commission | Horizon 2020 European Union funding for Research & Innovation

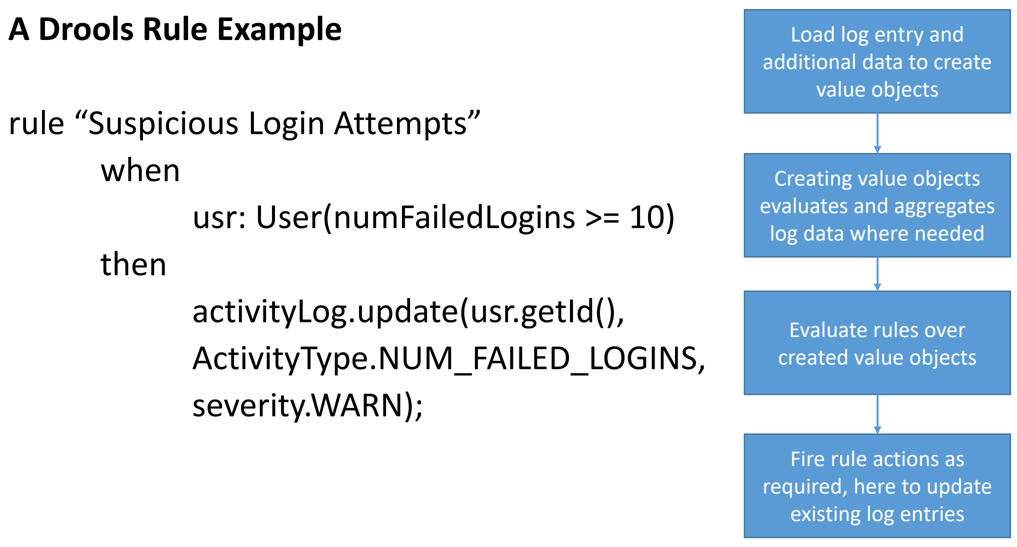
The main issue of interest in the Privacy Controller component are the Analytics Processor modules that carry out the analyses of activity log and other necessary data in the platform. The Privacy Controller integrates three different mechanisms that can be applied in order to analyse incoming and stored log data: a rule engine, an ontological reasoning module and an open source stream mining machine learning component.

The Privacy Controller analytics components all need to be configurable so that their rule sets, semantic models and learned models can be replaced and/or modified for different legal, ethics, privacy and organisational requirements. The Privacy Controller support this by externalising relevant models into file structures that can be edited easily in order to adapt the system to specific analysis requirements.

5.2 Rule Processing

A rule engine-based analytics processor module has been implemented using the well-known Drools Expert rule processing engine[4]. This rule engine is well-suited in order to cover analytics that evaluate low to medium complexity rule expressions (as opposed to more complex reasoning tasks). Custom rule sets can be defined and loaded on the fly for specific domain use cases.

Figure 5 depicts a typical log data processing rule that applies to general log data analytics and illustrates the overall principle of how Drools Expert rules are evaluated[5].



A Drools Rule Example

```
rule “Suspicious Login Attempts”
        when
                usr: User(numFailedLogins >= 10)
        then
                activityLog.update(usr.getId(),
                ActivityType.NUM_FAILED_LOGINS,
                severity.WARN);
```

Load log entry and additional data to create value objects
Creating value objects evaluates and aggregates log data where needed
Evaluate rules over created value objects
Fire rule actions as required, here to update existing log entries

Figure 5: Login Rule Example

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

[5] Please note that the rule descriptions in the provided examples are conceptual and do not represent the actual code used to define a Drools Expert rule.

Below describes a more specifically targeted rule definition, where a rule is defined in order to monitor the frequency with which face match requests are submitted to the system.

A Drools Rule Example

```
rule "Very frequent use of face matching"
        when
                inv : Investigation(numFaceMatchRequests > 100)
                Investigation(numUsers < 5)
        then
                activityLog.update(inv.getId(),
                ActivityType.FREQFACEMATCHING),
                severity.WARN);
                notifier.notify(inv.getOwner().getId(),
                ActivityType.freqFaceMatching);
```

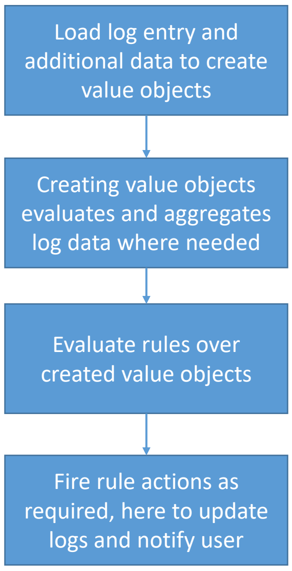

Load log entry and additional data to create value objects
Creating value objects evaluates and aggregates log data where needed
Evaluate rules over created value objects
Fire rule actions as required, here to update logs and notify user

Figure 6: System Analytics Usage Frequency Rule

Both cases illustrate the identical processing mechanism (shown in the blue boxes on the right) and the typical rule head -> rule tail structure of the rules defined using the Drools Expert rule system.

The rule tail in rules defined for the SPIRIT PEL subsystem prototype focus on adding information to display in the auditor log viewer, generating composite events that are displayed as output from analyses and on modifying alert levels for log database entries when rules fire for specific potentially suspicious activities.

5.3 Ontology-Based Reasoning

An ontology-based semantic reasoning system is a system that uses an ontological model and a reasoning system in order to evaluate data given the available model. Like the Drools Expert rule engine, ontology-based reasoners belong to the family of semantic reasoners. Semantic reasoners model data in in a way such that they can be evaluated in order to make logical inferences using the model, an appropriate evaluation mechanism and assertions such as observations (as described above for rule processing) or more generally logical axioms.

Using ontology models in order to evaluate log data in terms of compliance is in principle appealing, because significant efforts towards representing legal environments in an ontological semantic web infrastructure have been and continue to be undertaken in the legal domain (see e.g. [CAS16] for an overview and numerous examples). This means that it may be possible to leverage existing legal ontologies to represent legal requirements. The means for how to achieve this is to create a "low-level" ontology model that can relate the

– quite specific and technical – data gathered within the system with the high-level concepts defined in legal ontologies.

For the purpose of the SPIRIT project and the work in the context of the Project ethics sandbox, the Data Privacy Vocabulary DPV [PAN21][6] was selected in order to demonstrate how it may be practically used to derive a locally useable ontology for privacy protection.

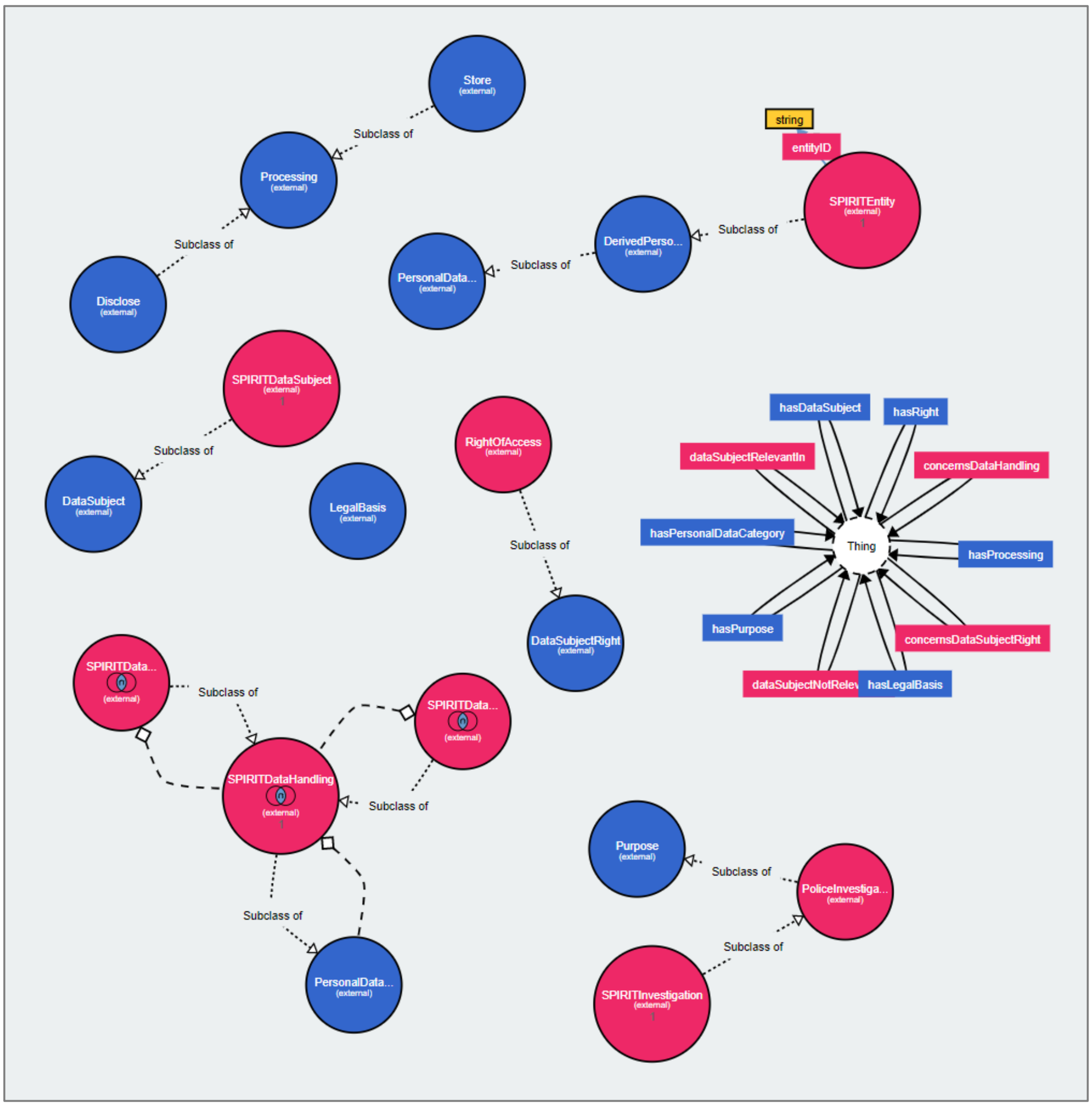


Figure 7: Example Data Integration Ontology

[6] See https://w3c.github.io/dpv/dpv/ for the draft vocabulary document

The developed practical ontology model describes concepts including entities and investigations as they are represented in the SPIRIT Mediator and connects those concepts with elements such as a data subject and data handling concepts as they are defined in the DPV. Figure 7 depicts a basic example of such a data integration ontology for this purpose.

In terms of data processing using such a model, several options can be considered in SPIRIT. Within the SPIRIT PEL subsystem, the open source semantic web framework Apache Jena[7] was integrated into the system in order to enable general reasoning support. Jena was configured so that reasoning can be triggered using Drools rules and so that the results generated via Jena reasoning could then be used to trigger rule tails as defined in Drools rules.

A second approach available in SPIRIT is to use the SPARQL [PRU08] query format to formulate reasoning as a query that can be executed against the SPIRIT Mediator RDF model. Figure 8 depicts an example SPARQL query that retrieves data stored about a user for review.

```
SELECT ?inv ?cat ?ID
   WHERE { ?x a :SPIRITDataHandling.
      ?x :hasPersonalDataCategory ?cat.
      ?cat a :SPIRITEntity.
      ?cat :entityID ?ID.
      ?x :hasPurpose ?inv.
      ?inv a :SPIRITInvestigation.
      ?x :hasDataSubject ?pers.
      ?pers :dataSubjectNotRelevantIn ?inv}
```

Figure 8: Example SPARQL Query

A mechanism to execute such SPARQL queries has also been implemented as an Analytics Processor that can be called from within a Drools rule analogously to the previously described approach for trigger Jena reasoning.

This second approach is appealing in the context of SPIRIT insofar that it uses the already existing infrastructure for semantic reasoning. From a log analysis perspective it is useful for evaluating data from the SPIRIT Mediator, whereas the Apache Jena integration can be applied to the activity log data, which is not accessible via SPARQL as the logging database cannot be queried via SPARQL.

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

5.4 Machine Learning Integration

The analytics approaches described in the previous subsections are concerned with methods that require the explicit formulation or modelling of domain data or explicit modelling and description of the domain including activities to be monitored. While it is possible to account for unexpected combinations of these activities using the described systems, these solutions are not well-suited to identify unexpected issues that have not explicitly been considered and defined/modelled appropriately.

Statistical machine learning techniques can be used in order to identify patterns in data, and from those patterns also identify unusual "outliers" in data that are presented to a suitably trained machine learning algorithm. This means that using a machine learning Analytics Processor would be potentially useful and complementary to the previously described processors.

Within the scope of SPIRIT however, it is not possible to realistically integrate and test or demonstrate a machine learning component due to the lack of available data and our lack of ability to gather data to use for development purposes for our law enforcement user partners.

We have nevertheless carried out a basic integration of a well-known existing as a basic provision for future work and exploitation of the system with minimal personnel effort. To this end we have implemented an interface to Apache Spark[8], which is an environment for statistical data processing that includes a stream processing pipeline which can be applied to data such as the ones envisioned for SPIRIT activity logging.

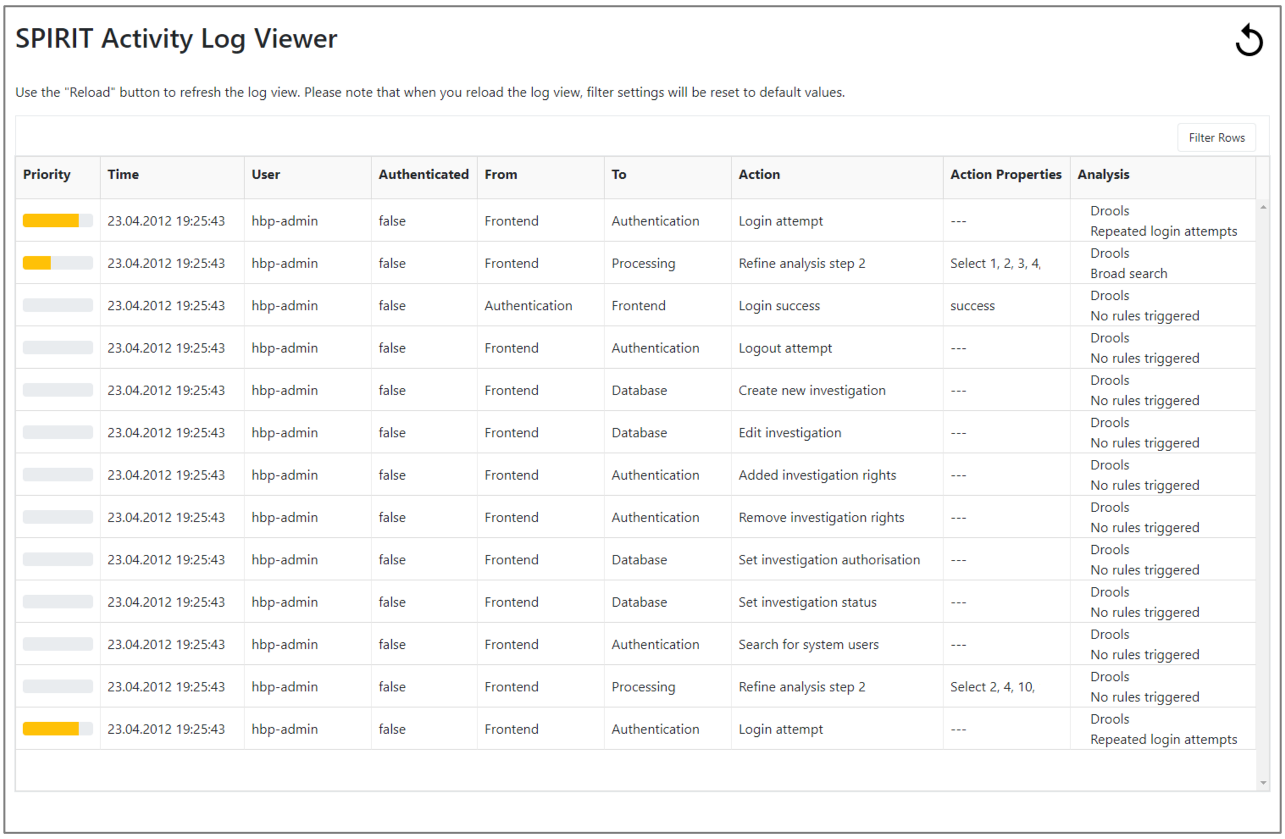
This integration enables us to show that a technical integration of machine learning analytics is feasible and to demonstrate this using simple toy examples.

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

6 User Interface Components

This section briefly introduces the user interface component that has been implemented for activity logging. The user interface components shown and described in this section have been implemented as stand-alone components in order to enable separation of concerns between typical system users and auditors as well as in order to enable concurrent user interface development by multiple parties in the project.

The Activity Log Viewer user interface element is a component that is used in order to view activity log entries for a system installation. Figure 9 presents a screenshot of the log viewer. The log viewer supports viewing the gathered log data as well as filtering log data using various filter criteria, viewing additional information on log data and filtering and viewing log data by review priority levels.



SPIRIT Activity Log Viewer

Use the "Reload" button to refresh the log view. Please note that when you reload the log view, filter settings will be reset to default values.

Filter Rows

Priority	Time	User	Authenticated	From	To	Action	Action Properties	Analysis
🟨	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Login attempt	---	Drools Repeated login attempts
🟨	23.04.2012 19:25:43	hbp-admin	false	Frontend	Processing	Refine analysis step 2	Select 1, 2, 3, 4,	Drools Broad search
	23.04.2012 19:25:43	hbp-admin	false	Authentication	Frontend	Login success	success	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Logout attempt	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Database	Create new investigation	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Database	Edit investigation	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Added investigation rights	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Remove investigation rights	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Database	Set investigation authorisation	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Database	Set investigation status	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Search for system users	---	Drools No rules triggered
	23.04.2012 19:25:43	hbp-admin	false	Frontend	Processing	Refine analysis step 2	Select 2, 4, 10,	Drools No rules triggered
🟨	23.04.2012 19:25:43	hbp-admin	false	Frontend	Authentication	Login attempt	---	Drools Repeated login attempts

Figure 9: Activity Log Viewer

The Privacy Controller Management user interface lets users manage basic elements of the Privacy Controller configuration through a web interface. The web interface allows users to enable and disable analytics components as well as to enable and disable specific Drools Expert analytics rules individually.

Horizon 2020
European Union funding
for Research & Innovation

7 Output Processing

This section presents use cases, workflows and example options for output processing within the scope of the SPIRIT project. Since the system specification for the SPIRIT red thread did not include integration of these output processing methods, they have not been integrated into the system prototype.

7.1 Use Cases

In SPIRIT, output processing refers to modifying the output generated by or the data used by a system component after it has been processed. The main group of target use cases for such activities is privacy protection in situations where this is necessary. We briefly describe two use cases:

- UC1: Selective anonymisation of source data prior to displaying it to end users of the investigator workflow. This involves blurring faces of persons on images unless the system end user has been authorized to access a particular set of personal data.

- UC2: Selective anonymization of source data prior to exporting it or presenting it to users not using the investigator workflow and user interface. An example for this would be auditor access for a random sampling review of body-worn camera footage recorded by police officers. In this instance, reviewers should not be able to identify persons visually or by direct identifiers such as names, so that all of those should be anonymised.

UC1 is an example for a generally applicable integration of output processing where all output would be anonymised unless a user is evaluated as being permitted to view personal information concerning a specific individual. In practice this is not typically applicable in the context of the work of the LEA partners in the project. UC2 is a real-world use case that has been referenced by LEA partners in the project but was not envisioned as a distinct part of the SPIRIT workflow.

7.2 Processing Flow

The envisioned processing flow for UC1 and UC2 assumes that as a baseline, all personally identifiable data for users from all source modalities (incl. the database) is sent to the user interface in anonymised format where the anonymised data is sent in an encrypted format with a separate encryption key for each relevant identity as well as a single encryption key for unidentified persons (e.g. background persons) in imagery or similar sources.

The end user can the request access to personal user information either explicitly by a user or implicitly by means of his access rights as an investigator. This request is sent to the Privacy Controller, which manages decryption keys for the encrypted information and provides decryption keys if the relevant authorisation (defined via Drools Expert rules) permits it. The user interface can then use the provided decryption key to decrypt personal data. This process is also illustrated in the figure below.

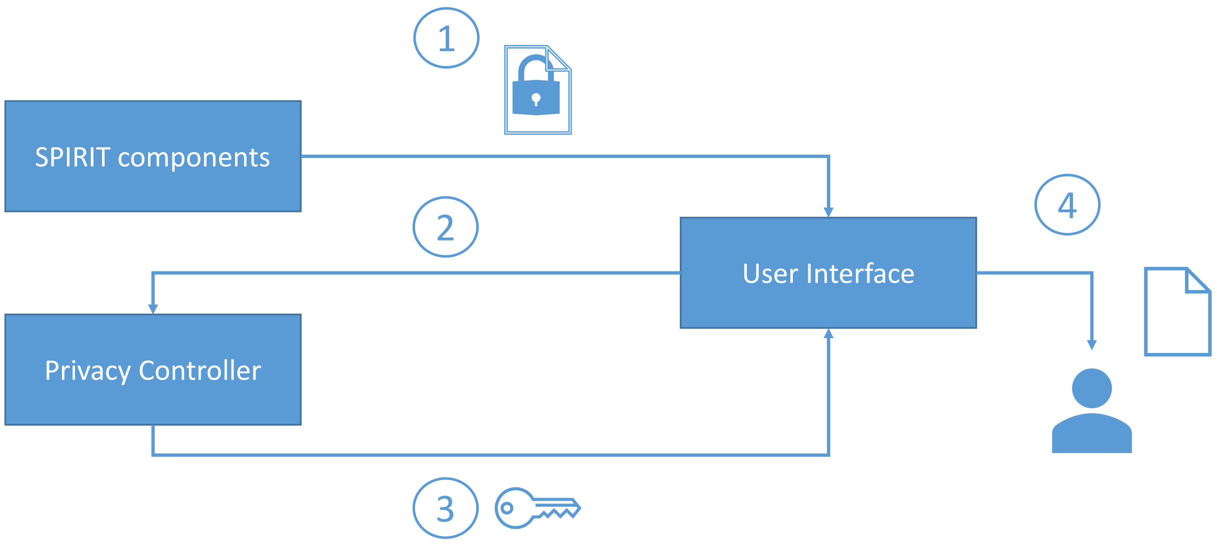


Figure 10: Envisioned Output Processing Flow

7.3 Output Anonymisation and Encryption/Decryption

Conceptually, the encryption of personally identifiable data that is transmitted as text data is considered straightforward, because the platform identifies potentially personally identifiable information as part of the analysis process and can thus easily apply encryption algorithms to the data (we ignore performance implications for this conceptual view). We discuss video anonymization here specifically as a more complex practical use case for the described approach.

The video processing services for face extraction (Face Extraction Service – FES) gathers metadata about face positions at each frame (discrete time stamp) in a video. Although we do not store this detailed information in the knowledge base, we store a detailed meta data file in the binary storage and link it to the face entities.

This enables us to selectively identify regions in the video that are linked to a specific entity and apply a post processing step to the video as required by UC1 as well as by the visual part of UC2. We theoretically describe two approaches to implement UC1 that we've experimented with during the very early stage of SPIRIT.

7.3.1 Destructive (Selective) Anonymisation

The simplest approach is to make all face regions in a video unrecognisable base on the extracted meta data. However, the technologies developed in SPIRIT (Face Matching Service – FMS) also allow for a selective approach: Based on selected face entities (references) we can render all face regions of a video unrecognisable *that match these entities*. This process can easily be turned around by making all regions recognisable that *do not match the references*. The latter might be the more realistic used case considering the reference being the face of a suspect and applying anonymisation to all non-related faces.

The technical process for the destructive approach is quite simple. The original video is decoded, and a simple images processing filter is applied to the face regions. The simplest and most irreversible method is to is just change each pixel in the region to black. Blurring or reordering in blocks may be unsafe since depending on the video resolution a reconstruction may be possible, or a human could still be able to identify a person at a different zoom level. In Figure 11 we give two examples where we used to reference entities (top left) and matched them in the different frames of a video. For illustration we applied a “set to black” filter as well as a rough “pixelation”.

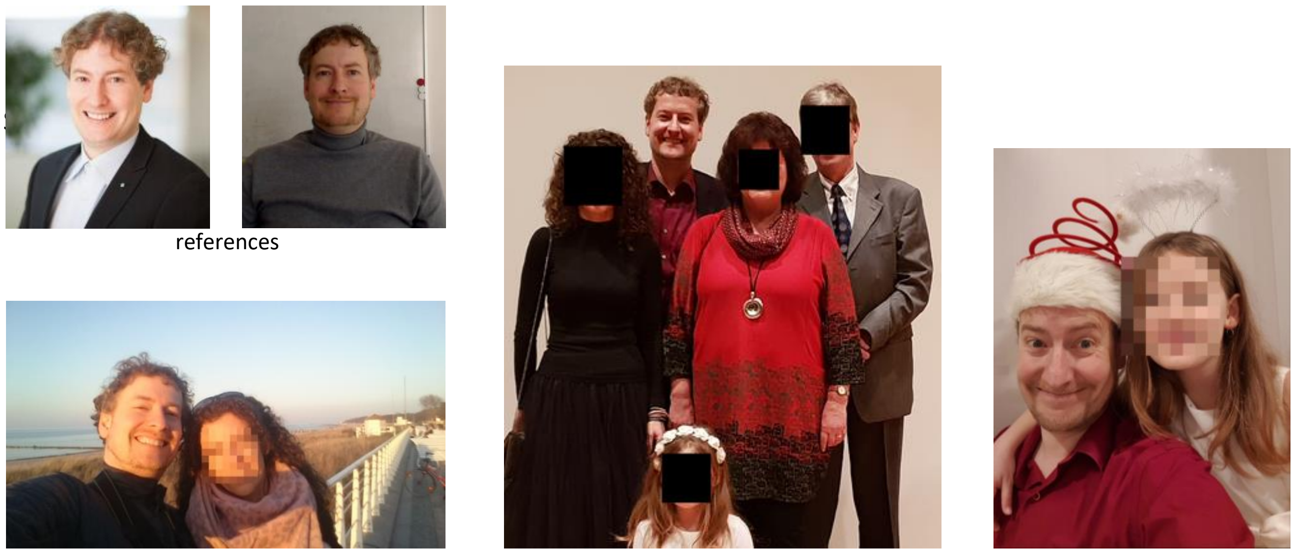


Figure 11: Example of a Selective Destructive Face Anonymisation.

7.3.2 Selective Anonymisation Via Face Encryption

We also investigated less destructive approaches that allow to reconstruction specific faces regions given the proper permission. In an experimental setup we encrypted the face regions with a standard symmetric encryption algorithm to be able to decode the face later. However, this requires a modification to the video codec interna. The encrypted video can be played by any standard player showing noise where encrypted faces

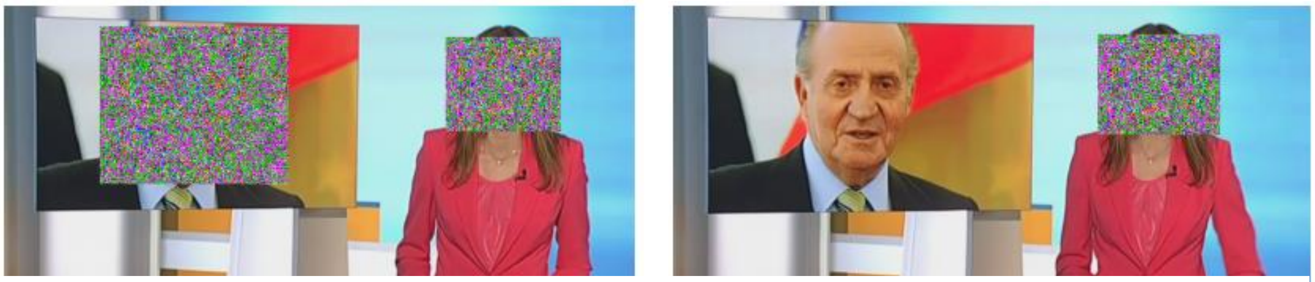


Figure 12: Examples for Selective Face Encryption

appear (see Figure 12 on the left). However, to be able to selectively decrypt a face (see Figure 12 right) a special player is required.

7.3.3 Risks & Challenges

No matter which technology (destructive/encryption) we use, the degree of anonymisation depends on the quality of the face detection and tracking algorithms (it might miss faces). Furthermore, the selection of faces via references depends on the accuracy of the matching process. Algorithms never achieve 100% accuracy for all kinds of material and the user must be aware of it. One approach would be to use the confidence levels of the detectors and matchers and setting a very narrow acceptance threshold ensuring that videos get flagged for human intervention where the detection or matching confidence is too low[9].

Figure 13 shows three examples. The red circles mark false detections. These are the worst-case scenarios since when we miss to detect a face, we can neither anonymise it

nor prevent from being anonymised. The orange circle shows a wrong anonymisation due to a missed match with the reference. In this scenario the false match does not weaken the level of anonymisation since more faces are anonymise that needed. However, it may become an issue when the face is needed visible in the context of an investigation and a destructive anonymisation technique was applied.

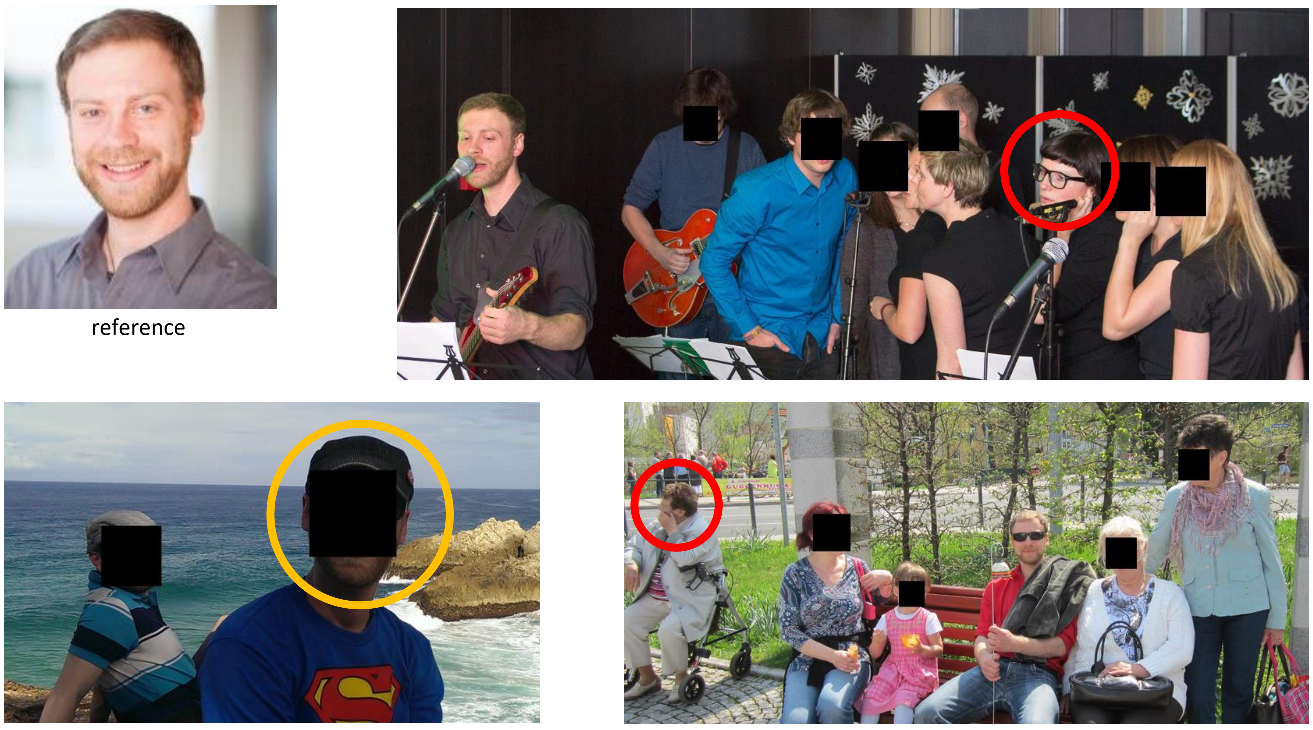


Figure 13 False Detections and False Match During Face Anonymisation

[9] Obviously, for missed detections there is no confidence available. In a video scenario we can still exploit the temporal properties of a face track. When a track of a person has short interrupts there is a high likelihood that the face detector missed the face in some frames, e.g., due to occlusions. This information can be used to flag the issue or even to try to interpolate the missing regions.

7.4 Conclusion

The investigation of anonymisation and reversible anonymisation in images as the more complex of the investigated media modalities shows the extent to which automated anonymisation and reversible anonymisation are feasible within SPIRIT, and it serves as a reminder than fully automated anonymisation processes are limited by the performance of the underlying algorithms that identify individuals and personally identifiable information.

The process approach for "unlockable anonymisation" described in Section 7.2 shows a simple method by which such an approach could be implemented. In practical terms, a number of barriers would need to be overcome in order to implement this approach:

1. Performance impacts would need to be addressed both for the encryption/decryption and the generation and retrieval of the respective keys.
2. System integration impacts would need to be addressed. These would likely significantly increase the complexity of the developed system, since all components that provide user identifiable data to the end user interface would need to implement it. Components from which large amounts of data are retrieved would incur an increasing performance penalty if identities would need to be anonymized as a default.
3. The system usability would need consideration in terms of how to present anonymised data and how to offer decryption functionalities. Furthermore, the implementation effort for user interface development would significantly increase given that multimedia presentation methods would need to support decryption and display of encrypted and decrypted media.

These barriers indicate that the implementation of a system such as the one we have outlined in this section in full would require substantial development effort and would impact the hardware requirements for deployment. From this perspective, implementations for UC1 would only become feasible once there is a clear need for such solutions from the potential customer base. UC2 could be implemented more feasibly in a smaller, potentially stand-alone tool to facilitate the reversible anonymisation of body-worn camera content only beyond the scope of SPIRIT.

European Commission | Horizon 2020 European Union funding for Research & Innovation

8 Conclusion

This brief section concludes the deliverable document.

8.1 Summary

This document is part of the deliverable D9.8; the software system that is described in this deliverable is available on the SPIRIT Gitlab repository (for those parts of the software that require source code availability for integration) and as Docker images on the project-internal Docker repository (as ready-to-run images).

8.2 Outlook

The work presented in this deliverable provides an overview of the developed prototype implementation of the PEL subsystem for SPIRIT. As we describe in multiple sections of this document, the subsystem is highly configurable but also relies of such configurations and specifications in order to be a useful tool for activity logging and activity log analysis.

Future work should, also with a view towards commercialisation, focus on developing a standard basic configuration covering all typical security requirements of LEA (or other domain) customers and should make use of more advanced analysis functionalities such as statistical anomaly detection to augment this. Commercialisation within SPIRIT and beyond would furthermore be aided if standard interfaces to typically used third-party tools could be provided as part of the subsystem.

References

[CAS16] Casanovas, P., Palmirani, M., Peroni, S., van Engers, T. and F. Vitali: Special Issue on the Semantic Web for the Legal Domain. Editorial: The Next Step. In: *Semantic Web Journal*, Vol. 7, No. 3, 2016, 213-227.

[PAN21] H. Pandit (ed.): Data Privacy Vocabulary (DPV), version 2.0. https://w3c.github.io/dpv/dpv/, last accessed 29.07.2021, 14:16.

[PRU08] E. Prud'hommeaux: SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. https://www.w3.org/TR/rdf-sparql-query/, last accessed 29.07.2021, 14:16.

[SNO04] Snodgrass, R.; Yao, S. and C. Collberg: Tamper Detection in Audit Logs. In: Proceedings of the VLDB '04 Thirteenth International Conference on Very Large Data Bases, 2004, 504-515.

Appendix A: Developer Documentation for Authentication and Authorisation

Appendix A is omitted in the public availability release of this deliverable. This is because the document contains confidential information which could be used in order to target attacks on the authentication and authorisation subsystem, thus improving chances of compromising system integrity.

The appendix will be made available to project development partners and Commission services upon request.

Appendix B: Developer Documentation for Customisation of RabbitMQ

Appendix B is omitted in the public availability release of this deliverable. This is because the document contains confidential information on the specific configuration of the RabbitMQ messaging system used, which could be used in order to carry out targeted attacks in particular concerning attacks to degrade system response times. Hence this appendix has been omitted from the public release of this deliverable.

The appendix will be made available to project development partners and Commission services upon request.

Appendix C: Developer Documentation for Activity Logging API

Scope

This guide documents the functionalities of the activity logging subsystems in SPIRIT. The subsystem is provided as part of the spirit-syncope image; it requires a running spirit-postgresql image instance.

Naming Conventions

The logging system requires strict adhere to naming conventions in particular for two key instances:

1. Naming of system components
2. Naming of actions carried out

We provide reference tables for both of these groups of instances below. The Component identity is usually the same as the component docker image name it is deployed with; exceptions may be made when a single docker image contains numerous functionalities that are sufficiently distinct to warrant identifying them separately. The name should be used without the formal repository address unless strictly required to differentiate it from other modules within the project.

Please note: the name should include the system version of the component used at the time of logging. This is in order to enable us to identify the system version that was used that the time of logging an activity.

Please note: For practical purposes, this notion is of note only for the source of a message, since the sender will be able to easily add the correct version of the sending component as part of a message.

Table 5: Component to identifier mapping

Component	*Component Identifier*
User Interface	ui-service:*[version]*
Syncope	spirit-syncope:*[version]*
API Gateway	api-gateway:*[version]*
Scheduler Service	scheduler-service-basic:*[version]*
Mediator Service	mediator-service:*[version]*

Additional component identifiers should be defined in the same manner as outlined in the table above. The table will be periodically updated in order to reflect the identifiers used in the project. Please note that we may begin rejecting messages that validate these identifier requirements if it is deemed necessary to enforce correct identifier use.

Table 6: Action to identifier mapping

Action Reference	*Component*	*Action*	*Action Identifier*
User Interface, Login & Logout			
001	ui-service	User submits login information via user interface	login-submit
002	ui-service	User is informed that login has been successful	login-success
003	ui-service	User is informed that login has not been successful	login-failure
004	ui-service	User submits logout button	logout-submit
User Interface, Investigation Handling			
005	ui-service	User submits data for investigation to be created	investigation-create
006	ui-service	User submits data for investigation to be edited	investigation-edit
007	ui-service	User submits information to add a user to an investigation	investigation-useradd
008	ui-service	User submits information to remove a user from an investigation	investigation-userdelete
009	ui-service	User submits information to change rights of a user	investigation-rightsedit
010	ui-service	User submits information on authorisation form	investigation-authform

011	ui-service	User sets an investigation to inactive	investigation-status-inactive
012	ui-service	User sets an investigation to active	investigation-status-active
Users			
013	ui-service	User searches for a user in the system	users-search
Analysis			
014	ui-service	User submits analysis process	analysis-submit
015	ui-service	User submits refined search intermediate refinement	refinement-stepone
016	ui-service	User submits refined search final refinement	refinement-steptwo

The table of actions is defined by the WP9 team. The Action Reference field is not to be used in code, instead the Action Identifier is to be used. The Action Reference is to be used when discussing a specific action and never in code.

REST Interface

The REST interface for activity logging is NOT the default option for logging system activities. It should only be used when you have been instructed to use it for part or all of your logging activities, and then it should only be used for those activities for which you have instructed to use the REST interface.

The REST interface provides a Swagger online documentation to users, which is available at

http://localhost:9080/logger/swagger-ui.html

(assuming you are running an instance of the system on localhost).

The main method for submitting log data via the REST interface is the method at the /logger/log POST endpoint. Additional logging methods will provide convenience activity log submission endpoints as deemed necessary by the developers.

The /logger/log endpoint accepts a JSON message that used the following template at the time of writing. Please note that not all of the information in the format may be submitted by you. Elements that may not be submitted by you are indicated in *italics* in the table below. Elements that are mandatory (when available, see table entries for details) are indicated in **bold** typeface.

Table 7: General REST Endpoint JSON Logging Format

```
{
  "id": 0,
  "username": "string",
  "bearertoken": "string",
  "isValidated": false,
  "time": "string",
  "source": "string",
  "target": "string",
  "action": "string",
  "payload": [
    {
      "id": 0,
      "path": "string",
      "mode": "string",
      "body": "string"
    }
  ]
}
```

We explain each of the elements separately below.

Table 8: Class Activity Elements

Element	*Description*
id	Unique identifier for an activity log entry, will be assigned by the system and cannot be assigned by you when using a log endpoint
username	Username, should always be included in calls
bearertoken	Bearer token, should always be included in calls when it is available; can be omitted when it is not available (e.g. before authentication or in cases where a system component does not receive a bearer token)
isValidated	System property concerning user validation, any values submitted by users during creation will be reset to false, so should be omitted to save bandwidth
time	Time of the action in ISO8601 format as follows: “2019-07-29T15:26:26+00:00” with offset for a time zone used or “2019-07-29T15:26:26Z” when using UTC time; the time reported should be as close as possible to the time of the action
source	The source of the activity log entry. This is the component id of the system sending the information of concern in the entry.

target	The target of the activity log entry. This is the component id of the system receiving the information of concern in the entry. This may be omitted if there is no receiving system involved.
action	The action being reported as defined in the action table in this document
payload	A list of PayLoad elements that contain the message data that is being sent. The format of PayLoad messages is described in the table below. Each Activity may contain a list of PayLoad elements; where possible, the elements should be represented in order to occurrence. PayLoad elements are also used to represent data retrieved from the SPIRIT Mediator.
parameters	A hash map of parameter key/value pair String data that document component configuration parameters have used for processing. The parameters used should document all parameters settings that deviate from the default settings used for the component version that is used and referenced in the "source" element of this Activity element
output	A list of references to output generated by the system that is stored somewhere; can either be an internal path to a file or a URL referencing a specific request to the SPIRIT Platform database

The PayLoad class should be used in order to record the messages send by one component to another. Where applicable, the system should send copies of the actual messages sent in order to ensure complete logging coverage and in order to eliminate any need for interpreting messages for activity logging purposes. The PayLoad elements references above are provided in the following table:

Table 9: Class PayLoad Elements

Element	*Description*
id	Unique identifier for a PayLoad entry, will be assigned by the system and cannot be assigned by you when using a log endpoint
path	The path that you are accessing with a call; for a REST call this would be the URL used for the call. If a call is made in a different format, please identify the nearest equivalent to a path (e.g. database name:table name for a database update)
mode	The mode in which you are making a call; for a REST call this would be the method used (e.g. GET or POST). If a call is made in a different format, please try to identify the nearest equivalent to a mode, unless that information is already contained in the path or body, then leave mode empty.

body	The body of a message. For a REST call this would be the message body; for a RabbitMQ message this would be the content of the message sent to the relevant queue. In instances where copies of write submissions to the database are communicated, the "create" or "update" instruction should be included, but the values to be inserted do not need to be provided (while fields do). ***Please note:*** *all body information* ***must*** *be sent base64-encoded in order to ensure that valid JSON data is transmitted to the logger. The system will always base64-decode at the receiver end, so information sent as body must be base64 encoded in* ***all*** *cases.*

Annexes

Annex A: Cheat Sheet for User Interface Developers

Table 10: Action to identifier mapping

Action Reference	Related User Interface Action	username	bearertoken	time	source	target	action	payload
User Interface, Login & Logout								
001	User pushes the login button on the user interface	username given in login form	-	current time	ui-service :[version]	spirit-syncope	login-submit	-
002	UI shows user as logged in in the user interface	username	token	current time	ui-service :[version]	-	login-success	-
003	UI shows message about failed login to user	username	-	current time	ui-service :[version]	-	login-failure	-
004	User pushes the logout button on the user interface	username	token	current time	ui-service :[version]	spirit-syncope	logout-submit	-
User Interface, Investigation Handling								
005	User submits form for creating a new investigation	username	token	current time	ui-service :[version]	mediator-service	investigation-create	Message/s sent to the database
006	User submits form for editing a new investigation	username	token	current time	ui-service :[version]	mediator-service	investigation-edit	Message/s sent to the database

Horizon 2020
European Union funding
for Research & Innovation

007	User adds user to an investigation	username	token	current time	ui-service :[version]	mediator-service	investigation-useradd	Message/s sent to the database
008	User removes user from an investigation	username	token	current time	ui-service :[version]	mediator-service	investigation-userdelete	Message/s sent to the database
009	User submits information to change rights of a user	username	token	current time	ui-service :[version]	spirit-syncope	investigation-rightsedit	Message/s sent to Syncope
010	User submits authorisation form	username	token	current time	ui-service :[version]	mediator-service	investigation-authform	Message/s sent to the database
011	User sets an investigation to inactive	username	token	current time	ui-service :[version]	mediator-service	investigation-status-inactive	Message/s sent to the database
012	User sets an investigation to active	username	token	current time	ui-service :[version]	mediator-service	investigation-status-active	Message/s sent to the database
Users								
013	User searches for another user	username	token	current time	ui-service :[version]	spirit-syncope	users-search	Message/s sent to Syncope
Analysis								
014	User submits analysis configuration for processing	username	token	current time	ui-service :[version]	api-gateway[10]	analysis-submit	Message sent for processing
015	User submits selected subset of URLs for next refinement phase	username	token	current time	ui-service :[version]	api-gateway[1]	refinement-stepone	Message sent for processing
016	User submits selected subset of URLs from final refinement phase	username	token	current time	ui-service :[version]	api-gateway[1]	refinement-steptwo	Message sent for processing

[10] I am assuming that the communication with the refined search will be made through the API Gateway; if it is meant to work like that then please use api-gateway as the target here.