Co-inform

Context Matters,
Your Sources Too

Generic Co-Inform architecture – Version 2

D4.2

#ThinkCheckShare

Document Summary Information

Project Title:	Co-Inform: Co-Creating Misinformation-Resilient Societies		
Project Acronym:	Co-Inform	**Proposal Number:**	770302
Type of Action:	RIA (Research and Innovation Action)		
Start Date:	01/04/2018	**Duration:**	36 months
Project URL:	http://coinform.eu/		
Deliverable:	D4.2: Generic Co-Inform architecture – Version 2		
Version:	1.0		
Work Package:	WP4		
Submission date:	31/12/2018		
Nature:	P	**Dissemination Level:**	P
Lead Beneficiary:	Scytl Secure Electronic Voting, SA (SCYTL)		
Author(s):	Pau Julià, *Director of Security* (SCYTL)		
Contributions from:	Ronald Denaux (ESI) Tracie Farrell (OU) Cecilia Magnusson Sjöberg, Professor, LL.D (SU) Stefan Nenzén, Software Engineer (SU) Adrià Rodríguez-Pérez, Project Manager (SCYTL) David Salvador, Technical Manager (SCYTL) Syed Iftikhar Shah (IHU)		

The Co-inform project is co-funded by Horizon 2020 – the Framework Programme for Research and Innovation (2014-2020) H2020-SC6-CO-CREATION-2016-2017 (CO-CREATION FOR GROWTH AND INCLUSION).

Revision History

Version	Date	Change editor	Description
0.1	15/11/2018	SCYTL	Initial version
0.2	28/11/2018	ESI	Added contribution from ESI
0.3	30/11/2018	OU	Added contribution from OU
0.4	30/11/2018	SCYTL	Added contribution from SCYTL
1.0	11/12/2018	SCYTL	First draft complete version
1.1	15/12/2018	SU, IHU	First review
1.2	21/12/2018	SU	Final review and submission

Disclaimer

Copyright Message

Executive Summary

This report provides a second version of the architecture of the Co-Inform platform. It focuses on interoperability at business and information level, to be developed in an agile manner.

The system architecture of the Co-Inform project will be component-based and layered. Robustness, scalability and multi-user access support are important system characteristics. The architecture is designed based on general requirements from WP1 and its output will be a specification of the components of the system (coming from the research project partners in WP1 and WP3), of how they interact (e.g., flows of control and data), and specification of programmable interfaces so that different partners can build their components independently. In addition to the components from the technical partners, the architecture provides infrastructural components required in the system's design, some examples being a data store, security (access control), and scheduling of automated tasks.

This deliverable upgrades the details provided for the platform architecture in D4.1. It feeds from the latest technical developments from partners involved in WP3 and WP4.

This deliverable will be complemented in two additional iterations (D4.3, and D4.4), to be submitted respectively in M15, and M21. The upcoming versions will feed from the on-going results from WP1, WP2 and WP3.

Table of Contents

Table of Figures

1. Introduction

Misinformation online generates misperceptions. The speed and ease in which false news spread on social media have a massive impact on current affairs and policies. By bringing together a multidisciplinary team of researchers and experts in computer science, behavioural science, and sociology, Co-Inform aims at engaging all stakeholders in fighting misinformation by providing them with the tools to identify 'fake news' online, understand how they spread, and provide them with verified information.

To this end, Co-Inform will integrate its ICT tools and services (WP3) and policy encodings (WP2) to deliver a co-created misinformation resilience platform in the form of:

- A browser plugin to raise citizens' awareness of fully or partially misinforming content, of related fact-checking articles and corrective information, of average citizens' perceptions towards this content, and of key pro and against comments from fellow citizens.
- A dashboard for fact-checking journalists and policymakers, showing that misinformation was detected, where originated from, how and where it has spread and will spread in the near future, what's the current and predicted public perception, and what are the key comments about it from the public. The dashboard will also show the news articles or information that users requested to be fact-checked.

This deliverable describes the high-level architecture of the Co-Inform platform, including the interaction between its expected components.

This is a preliminary version of the architecture, which builds on the contents of D4.1 and feeds from the developments in the framework of WP3 and WP4. The goal of the document is to study different technical designs to accommodate the end user requirements that will be gathered in the framework of WP1.

The current document will be again reviewed, updated and detailed as required in order to comply with the set of requirements gathered with the platform's end-users in the course of the activities within WP1, with the management policies from WP2 and with the services developed in WP3. Specifically, it will be updated in two additional iterations (D4.3, and D4.4), to be submitted respectively in M15, and M21.

2. Co-Inform high-level architecture

2.1. Diagram

The following diagram illustrates the high-level architecture proposed for the Co-Inform platform:

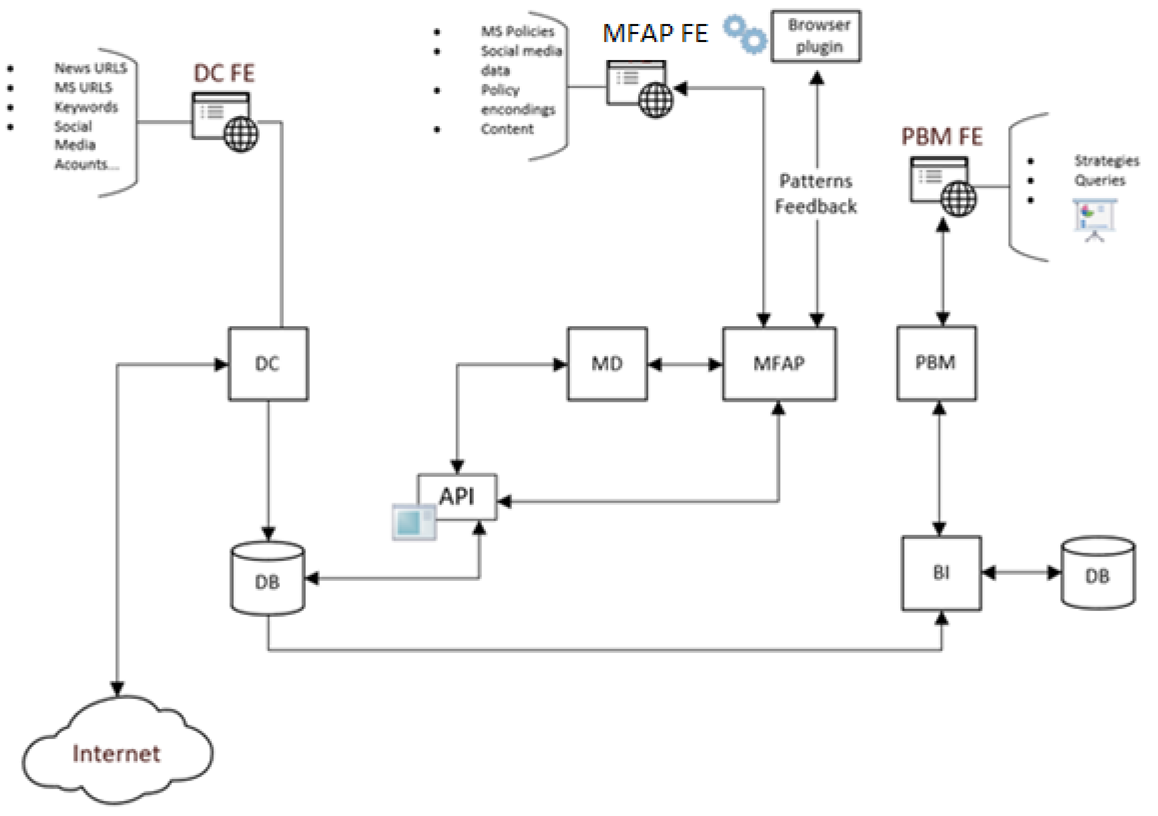

Figure 1 High Level Architecture

API: Application Programming Interface	**MFAP**: Misinformation Flow Analysis and Prediction
BI: Business Intelligence	**MFAP FE**: Misinformation Flow Analysis and Prediction Front End
DB: Database	**PBM**: Perception Behaviour Mining
DC: Data Collector	**PBM FE**: Perception Behaviour Mining Front End
DC FE: Data Collector Front End	
MD: Misinformation Detection	

2.2. Components' description

The following components will be needed to implement the Co-Inform architecture, based on the technical proposal submitted to the European Commission and the latest developments within WP3 and WP4.

Depending on the specific requirements gathered within the activities carried out in WP1, some of these components may be integrated into a single component or some of them could be split in other modules. These modules must be understood as a baseline to implement a more detailed infrastructure once these requirements are defined.

i. Data collector (DC)

The **Data Collector** (DC) is the module in charge of collecting the information that will be processed using ESI's industry-leading text analytics technology. Broadly speaking, this module will accept as an input a list of **sources** and **configurations** that it will use to start a text analysis process. After a short time, the found documents, along with the *analysis results,* will be placed in a **database**; the contents of which will be searchable and accessible to other modules via a document **API**.

Next, we explain the various terms, data structures and subcomponents in more detail.

The **Data Collector Front End** (DC FE) provides a way to specify and manage (i.e. add, remove, edit) sources and configurations to the DC. Although the types of supported sources have to be formally defined as part of pilot requirements in WP1, we envisage support for the following types of sources: websites to crawl, RSS feeds, social media accounts/pages, search keywords and off-line documents (e.g. PDFs, Word documents, etc.). Besides specifying the type of the source, the administrator will have to specify additional configuration parameters such as:

- Locator: this will typically be a URL to describe the site (e.g. https://snopes.com), rss feed (e.g. https://www.snopes.com/feed/), search engine (e.g. google, bing + keywords), location of the off-line documents.
- Collection: Since we will have multiple pilots, it makes sense to keep documents from each pilot in a separate 'section' of the database. This means that sources need to be bound to a specific collection/pilot. We assume that most sources will be specific to each pilot, although in some rare cases it may be possible that the same source will be used in multiple collections. The system may process these sources multiple times, although this could be avoided as part of the implementation of the data collector. Another reason why specifying a collection is useful is because we may want to perform special types of analyses on documents for a particular use case in the future (e.g. if we know that the use-case is migration, we may want to use a custom document classifier on these documents. Likewise, if we know that the use case is about a specific area in Sweden, we may want to provide use a text analysis service that has a more fine-grained knowledge of places, people and political issues in Sweden, rather than using a generic text analysis service).

- Frequency: how often should the source be re-crawled. Some types of sources are dynamic - they change every few hours (e.g. news feeds); while others are static (e.g. news articles).
- Crawl depth: in the case of websites to crawl, but also social media accounts and search keywords, it can be useful to not only collect the initial website and social media posts, but also to collect further pages/posts referenced by them.

The configuration options presented above are indicative. We will specify the list of configuration parameters based on requirements, implementation and usage. At the moment it is unclear whether the DC FE should be a graphical interface (GUI) or an API. An API is more flexible, as the graphical interface can be built on top of the API. It is also unclear who will be the stakeholder who will have access to this interface. In general, it makes sense to restrict access to this interface as it can have an impact on the number of documents that will be in the system, thus affecting possible system performance. An overly broad selection of sources can result in overly large collections of documents, which will make the overall system harder to use as there may be too many documents to explore and analyse.

The **Data Collector** submodule will use the sources and configurations specified using the DC FE to crawl the web, find documents and analyse them. As described above, some sources may be re-crawled at defined intervals. Figure 2 describes the working of this submodule in more detail and shows that it has two main functionalities: crawling and datamining.

- The crawler (Data Processing Engine in Fig. 2) processes the input document (e.g. an html page or PDF) and extracts its textual content and metadata (e.g. published date, author). A recent development that is relevant to Co-Inform is that many fact-checking articles are now including metadata (**https://schema.org/ClaimReview**)[1] that makes it easier for machines to gather the main claim being evaluated and provides a basic result for the fact-checking exercise. If this kind of sites will be included in the pilots, it may be necessary for the Data Collector component to include this kind of metadata as part of the extracted document (and eventually include this information as part of the database).
- The datamining step enriches the extracted document by automatically adding annotations such as categories (according to various generic taxonomies such as Intelligence, Crime and Terrorism), entities (places, people, organizations, domain specific), sentiment/emotion and relations (tuples or triples between entities and/or categories). Depending on the languages supported by the text analysers, the original document text may need to be translated. In particular, it may be necessary to translate all non-English documents into English (using third party machine translation services), to make it easier for researchers to evaluate and inspect the gathered documents.

[1] Fact-checking sites do this because it helps search engines such as Google or Bing to index these sites and highlight them as part of search results.

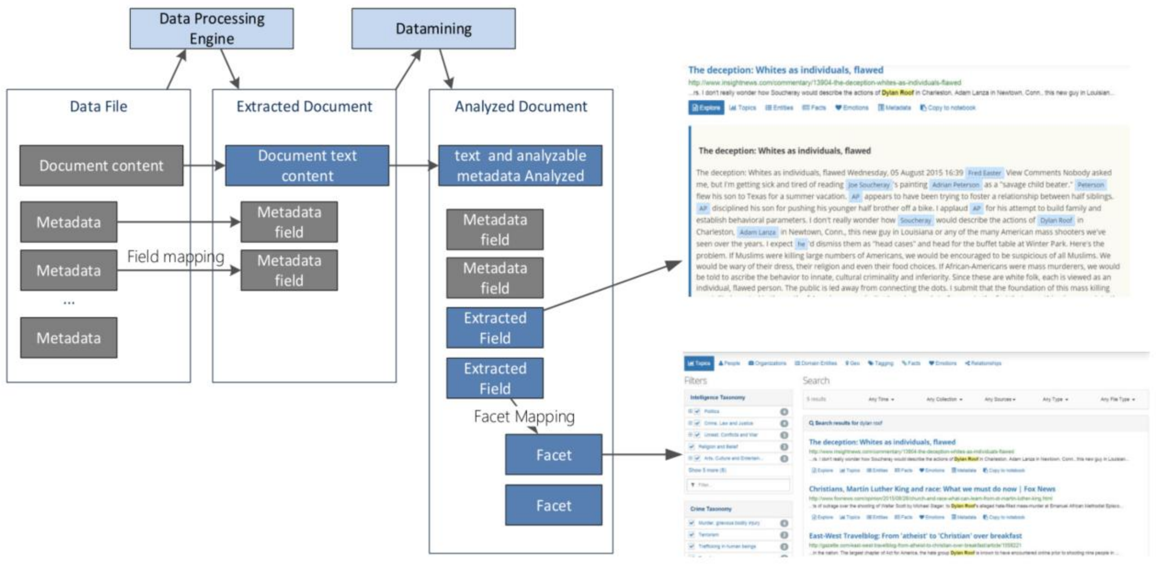

Figure 2 Crawling and Text Analysis

The **database** provides an API for accepting and updating analysed documents and stores them in representations optimized for storage and retrieval. It also provides a low-level API for retrieving analysed documents. Since this is an internal API, we do not specify it in further detail in this document and leave it as an implementation issue. The main constraints are that it must be able to store textual data (the content of the documents), metadata and extracted fields. Ideally, the internal API will allow for exploratory search of documents based on facets, since this allows exploring the dataset in an intuitive way (e.g. each search response lists the number of results which have a specific category or entity).

Finally, the database must be deployed in such a way that it safeguards in both a physical and legal manner any personal data. Therefore, the database will be compliant with requirements of information security as well as associated legal requirements laid down in GDPR and the project's Data Management Plan (D 7.1). As the project develops, data controllers and/or data processors (all project partners) will ensure to take appropriate measures in a case-by-case manner when faced with direct or indirect personal data following the provisions of GDPR article 4 et al. Further distinction will be made whenever data can be anonymised and therefore falling outside the scope of GDPR. Naturally, this needs to be continuously analysed in the consortium, since we need to balance:

- The fact that most sources are publicly available: i.e. does it make sense to anonymise mentions of Donald Trump in news articles? Similarly, mentions of users in Facebook or Twitter are already searchable.
- Informational requirements by downstream components. For example, the Misinformation Detection component may only work (or work much better) if the identity of authors of documents are known; hence in this case, anonymising this information will be counterproductive to the goals of the project.

The **document API** provides an interface available to other modules in the system for exploring and retrieving analysed documents. The API may be available as a web service (e.g. using REST), over a secure connection (e.g. requiring authentication and using secure HTTP). In a first phase, we envisage two main services:

- **Search**: will be used to search and explore documents in a particular collection. Since the main utility of this service is to search, by default, only a relatively small amount of information is returned for each document (for the full details on a document, the Document service can be used). This service will have input parameters:

    - Collection: a unique name or identifier
    - Keywords: a list of keywords that need to be matched against the text in the document. If deemed necessary, we may provide a simple query language similar to that used by well-known search engines in order to allow users to make some keywords mandatory, to allow for conjunction or disjunctions of keywords, etc.
    - Filters: a list of filters to narrow the set of results. These filters will typically be based on facets returned by previous search results. Filters will include categories, entities, dates (published, acquired), authors, sources, etc.
    - Pagination: since search results may be in the order of thousands of documents, clients may need to request the results in “pages”, where each page has a certain number of results.

    The output of this service will include fields like:

    - url: used to identify the document
    - metadata: a selection of the most relevant metadata; e.g. title, author, publisher, source, publishing date, published metadata.
    - Preview: a preview of the content (to avoid returning long documents)
    - Search result metadata: total number of results, pagination information, available facets.

- **Document**: will be used to retrieve detailed information about a small number of documents. The input will consist of one (or a few) mandatory document identifier (e.g. the url) and the collection name. The output will be a list of documents including the full textual content and all the metadata and extracted fields. Since this is not a search service, the response will not include search result metadata, nor a preview of the content, since the full content will be provided. If needed, we may provide further detailed information such as positions in the text for extracted fields (i.e. we may specify fields to include or exclude from the response in order for client components to be able to only retrieve the information that is required). Having said that, the default behaviour for this service will be to return as much information as possible for each document, since the client components can then extract or ignore information based on their needs.

To make integration with other components as easy as possible, we will aim at providing a specification of this API in a format that allows for easy client generation and testing, like

OpenAPI (**https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md**) and Swagger (**https://swagger.io/so lutions/api-design/**).

In terms of **performance**, the requirements again need to be specified as part of WP1. However, in technical terms, we assume that the Data Collector should be able to process documents in both batch mode and monitoring mode. The batch mode is useful at the beginning of a pilot, as many sources need to be ingested (depending on the quantity of sources and documents this may take minutes or hours). Once the initial set of sources has been processed, the monitoring mode keeps track of new documents at regular intervals. Analysis of these results should become available after a few minutes. Searching services should provide search results in mere seconds and quicker than that if they are to be used for interactive user-facing applications (which should be possible by using appropriate paging and search query parameters).

In terms of volume of documents, we expect that the conjunction of all pilots will not require storage of more than about ten million documents. If more documents are expected or required by downstream components or specific pilots, this will need to be specified in advance, since this may require special deployment plans to ensure good performance for such a volume of documents.

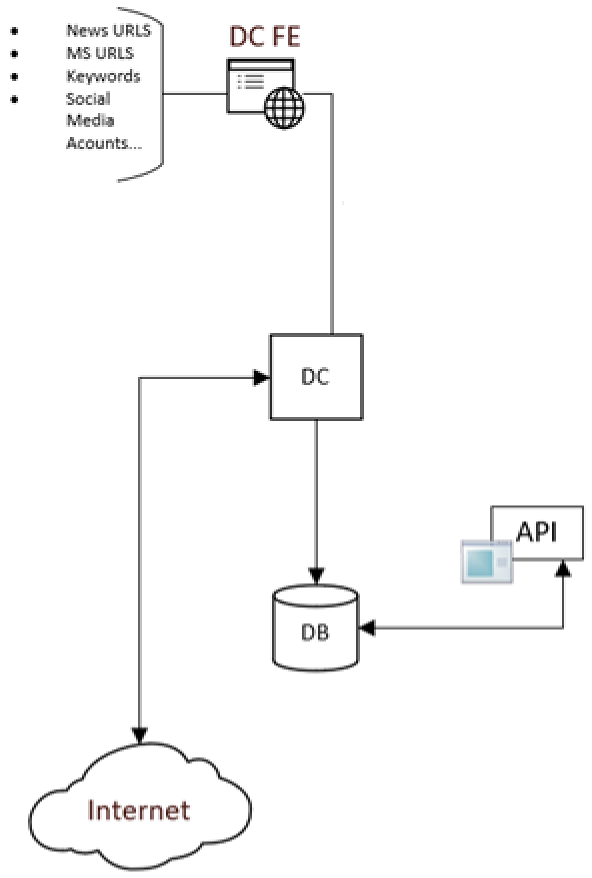

Figure 3 Data Collector

ii. Misinformation Detection (MD)

The **Misinformation Detection** (MD) module is developed as part of T3.2 in WP3 and aims at detecting misinformation using different methods. This will help managing misinforming content in large amounts of documents on social media or other information sources. As part of the work in T3.2, different models and services will be considered, such as an automated misinformation detection tool, and a dubious URL identifier service.

In general, most of the models developed by WP3 will require **annotated data** (although some models may be unsupervised) and will rely on **Machine Learning** (ML) models. Such data may be accessed from existing third-party data sources or from datasets shared with the Co-Inform consortium, as well datasets collected through the **Data Collector** (DC) module (T3.1). A typical MD service will need data that is different while it is trained (e.g., annotation labels) compared to when it is deployed for usage.

In most cases the input data required will need to contain the textual content (and media) of a given information that needs to be evaluated as well as associated metadata (e.g. content creator, sharing information, creation date, language, linked social network, etc.) as well as annotation labels if available that can be used for training the MD services (e.g. *information/misinformation* labels). This data can be collected from different information sources such as social media websites (e.g. Twitter, Facebook, etc.) or news websites. In general, this information is useful both for detecting the claims on social platforms (new ones and sharing known ones) and for analysing their spread and evolution.

Besides a particular content or claim, the following external information will be used as input such as:

- *Historical data about claims*: Fact-checking datasets can be used both for spotting and for predictive models. This type of data needs to contain information about particular claims (e.g. *who*, *what*, *where* and *when* a claim has been done). Also, some information from the fact-checking process of known fact-checker sources: a label indicating the type of claim (factual, fake, satire, ...), the author of the reviewing process;
- *Knowledge bases and ontological relations*: External knowledge bases (e.g. DBPedia, WikiData) can be also used for identifying semantic information related to particular documents and help study plausible relations in content;
- *Information network and/or social network*: Another important type of input that will be used is the network associated with the author of a claim or the claim information network (i.e. the interconnection between different claims). This data will be particularly relevant for bot discovery and the **Misinformation Flow Analysis and Prediction** (MFAP) module.

In general, the models will return probabilities of a given information (or URL or account) to be related to misinformative or informative content (or another relevant label). In the case of a dubious URL identifier, the service may be used for detecting new URLs containing misinformation or suggesting related URLs (e.g. similar historical content URLs or relevant fact-checking URLs).

The format of the output may be structured possibly following existing schemas such as the ClaimReview schema (http://schema.org/ClaimReview) or the Veracity ontology (http://socsem.open.ac.uk/ontologies/veracity/).

The trained MD services will follow standard API development practices by providing **RESTful web service** where the claim or other information that needs to be check is submitted to a service and a **JSON object** containing probabilities and labels is returned.

This module will be connected and will work together with the **Misinformation Flow Analysis and Prediction** (MFAP) module by providing the primary information necessary to analyse the flow of misinformation in social networks.

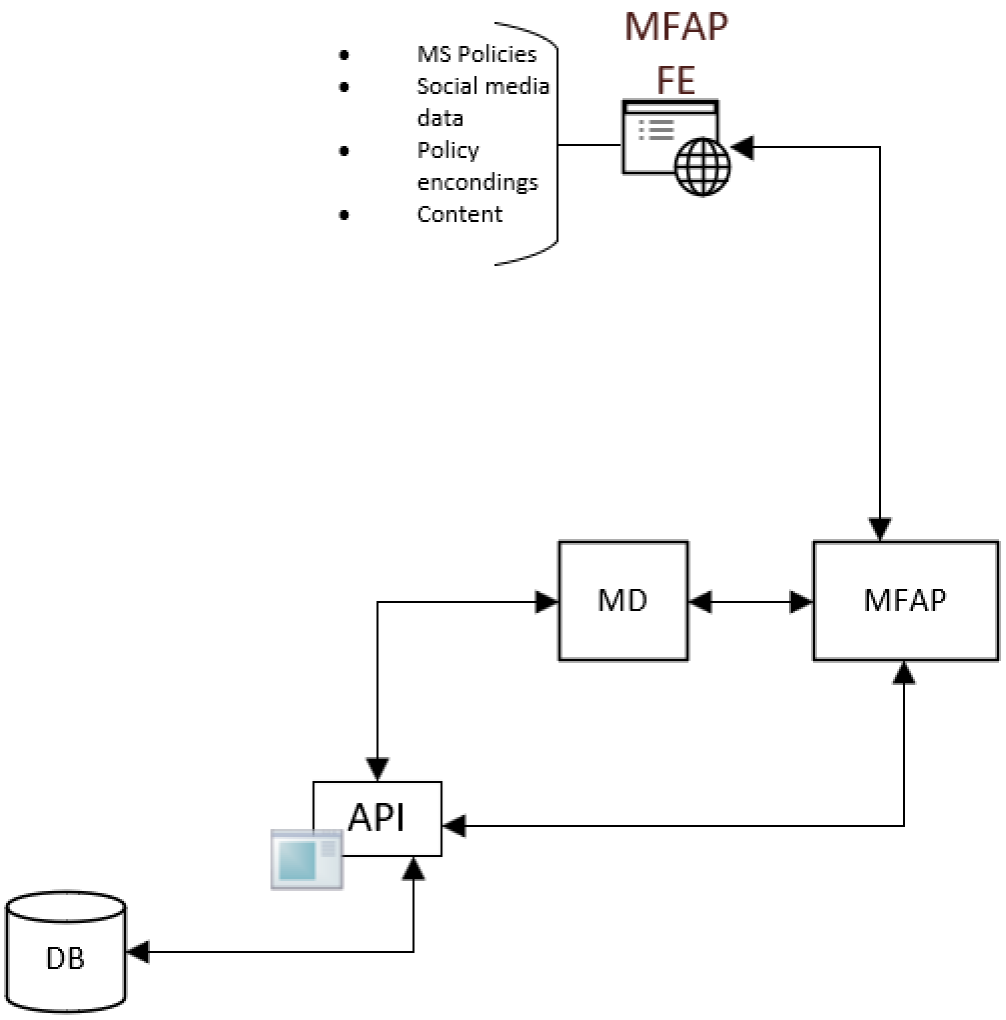

Figure 4 Misinformation Detection

iii. Misinformation Flow Analysis and Prediction (MFAP)

This **Misinformation Flow Analysis and Prediction** (MFAP) module will work together with the **Misinformation Detection** (MD) module using data similar to the one used by the MD module. The aim of this module is to determine patterns of different types of misinformation across the social networking platforms and to provide different metrics and measures associated with particular information flows in social networks in order to better understand how misinformation spread and evolves.

Similar to the previous module, the different services provided for the MFPA module will require historical data from different data sources including data collected by the **Data Collector** (DC) module. This module will require different types of input such as:

- *Misinformation management policies:* The WP2 policies can act as parameters that can regulate the behaviour of the model and/or inform the MFAP models' functionalities;
- *News articles* and *historical data:* Data collected over time such as the information collected by the DC module can be used for making misinformation predictions (via MD module);
- *Information network and/or social network data:* This data can be used to understand the *topology* and *typology* of the network that connects the different actors (publishers, accounts influenced by the content) identifiable in the historical data;
- *Misinformation analysis tools*: The MD module can be used for accessing important information about information nodes and actors and can be used to label some specific nodes in the social graph.

With this information available, the MFAP module should be able to track over time the diffusion of the claim instances, their evolution, reception, and amplification by different actors. The flow analysis will apply **Machine Learning** (ML) models that can be used to capture temporal features (such as the impact of a certain claim over time, or the related sentiment reaction to it) of the flow of misinformation.

This module will have several outputs, that will be made available to other modules. The results provided will be of different nature, such as:

- *Cascading patterns*: tracking the claims from their source and following them based on the historical data and social network information. These results can contain aggregated information that describe the spread of a certain claim, or a certain group of claims (identified e.g. by their topic, source, veracity...). Examples can be the number of shares/influenced users/likes/people talking about it over time, eventually describing how different groups of users/personas have been affected;
- *Predictors of misinformation flow*: given some seeds in input (such as claim, topic, source, veracity, sentiment), ML models could be used to predict the spread of the considered misinformation, giving insights like the number of people that will be influenced by it, and what their reactions would be.

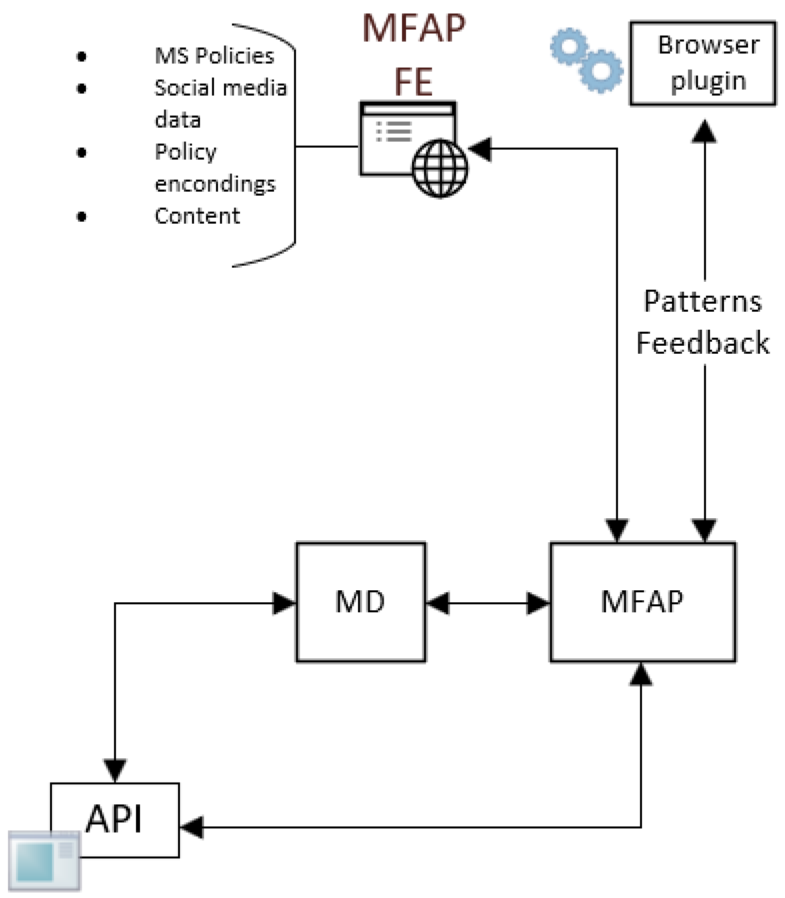

Figure 5 Misinformation Flow Analysis and Prediction

iv. Browser Plugin

The **Browser Plugin** will be the end user module (i.e. for social media users). It will analyse the end user navigation and together with the patterns provided by the **Misinformation Flow Analysis and Prediction** (MFAP) module will inform about the credibility of the information that they are accessing to.

For the plugin's architecture model three approaches can be taken. The approach selection will be made after and conditioned by the requirements extracted once the first workshops with end users are conducted in the framework of WP1.

The first approach is based on a *centralised architecture*. For every relevant website that the user loads, the plugin will send data to the MFAP module. This data will be the content of the website (mainly the text) and some metadata (website's URL, timestamp). Once this data is received by the MFAP, some anonymisation methods, such as IP removal, assignation of a transaction ID, etc. will be applied. Then, the data will be processed in order to diagnose whether it corresponds to misinformation or not. After the diagnostic, a response will be sent to the plugin which will show the user a **confidence level** on the website or post that is currently visiting. This model increases the complexity level of the MFAP by assigning more responsibility to it. More complex and robust detection rules can be generated but a higher system load (backend part) will be added with this approach. The anonymisation part is also a notable aspect in this architecture if good privacy practices want to be achieved.

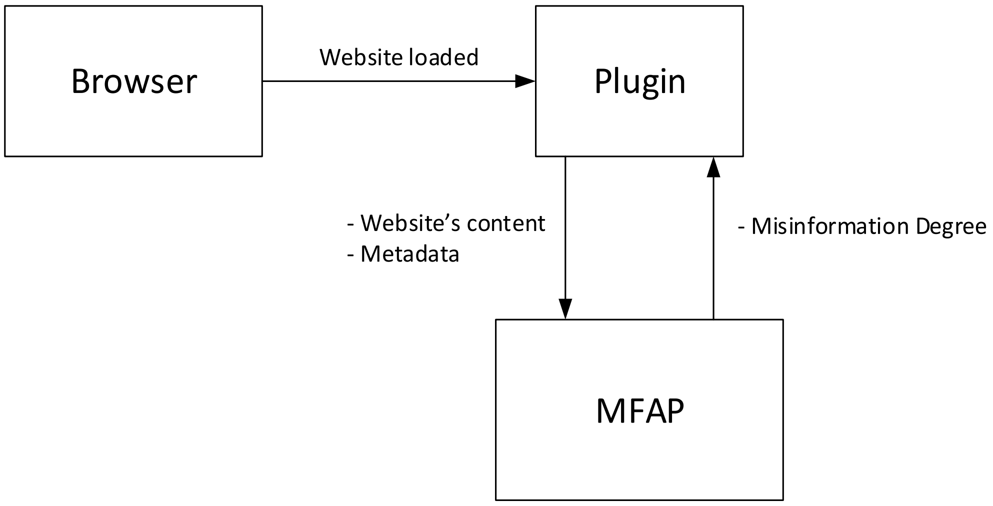

Figure 6 Centralised Architecture Model

The second approach is based on a *decentralised architecture*. The plugin will periodically download a set of rules from the MFAP. For every new website opened, the plugin will process the content of the website through a **rule engine** that will determine the credibility or misinformation degree of the information that the user is accessing to. This rule engine will be fed by the rules downloaded from the MFAP. This model decouples complexity and load from the MFAP and adds it to the browser plugin. It also does not require a data anonymization step, as all the data is locally processed. However, the rule set needs to be designed in a way that allow both for a fast processing, in terms of calculations required to achieve a result, and a fast distribution, in terms of data size.

The third approach is based on a *mixed architecture*. The plugin will have some set of rules provided by the MFAP. These rules will be less complex and less "smart" than the ones from the decentralised approach. Once a user loads a website, a first fast and less complex calculation will be made by the plugin using the rule engine with the rules set. In case of not being able to calculate a certain result, the plugin will send the website's content and some metadata to the MFAP for a more complex processing. With this approach, a first triage is performed locally by the browser plugin reducing the MFAP's load. In this way, the MFAP will contribute only to those cases where more complex rules need to be processed. However, a good balance between simple and more complex rules needs to be found for the system to be optimised.

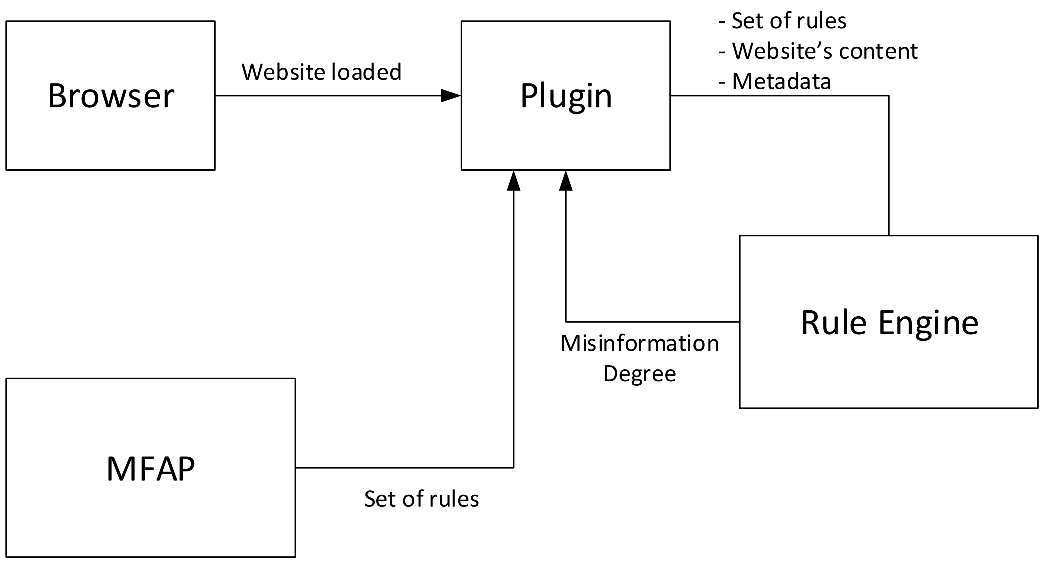

Figure 7 Decentralised Architecture Model

v. Perception Flow and Behaviour Mining (PBM)

Using the information stored into the database, the analysis and classification conducted by the **Misinformation Detection** (M**D)** module and the **Misinformation Flow Analysis and Prediction** (MFAP) module and using the statics and reactions collected by the **browser plugin**, this module will provide statistics on misinformation behaviours.

The core of this module is planned to be managed by a **Business Intelligence** service/application using an alternative database, which will be feed periodically by the central database (see section 2.2.i). The adoption of this approach will boost an increase in performance of the overall system.

This module will provide dashboards and statistics about the misinformation managed within the whole system. The specific parameters of each end-user will be inputted into the PBM module through a **web front end**.

User characteristics that could influence their engagement with misinformation might include age, culture, prior opinions, interests, exposure, etc. The PBM module will attempt to collect such information (e.g. from user's profiles, timeline analysis) to support the prediction analysis of misinformation flow. Spreading and acceptance or rejection of misinformation can be analysed (e.g. using **opinion mining**) to gauge the user's behaviour towards a particular piece of misinformation, and how this behaviour changes (or does not change) after an intervention (which will be provided by WP5) is executed. To this end, services for extracting user characteristics and opinion will be required, which would take input data such as: user profile, user posts (timeline), and off-the-shelf opinion mining methods.

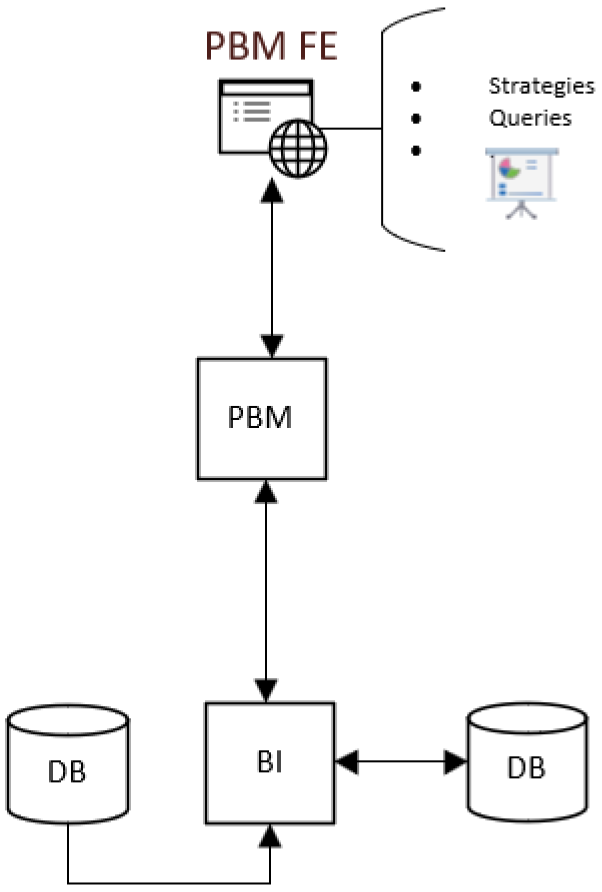

Figure 8 Perception and Behaviour Mining

2.3. Data Flow

The following diagram depicts the data flows between the different components:

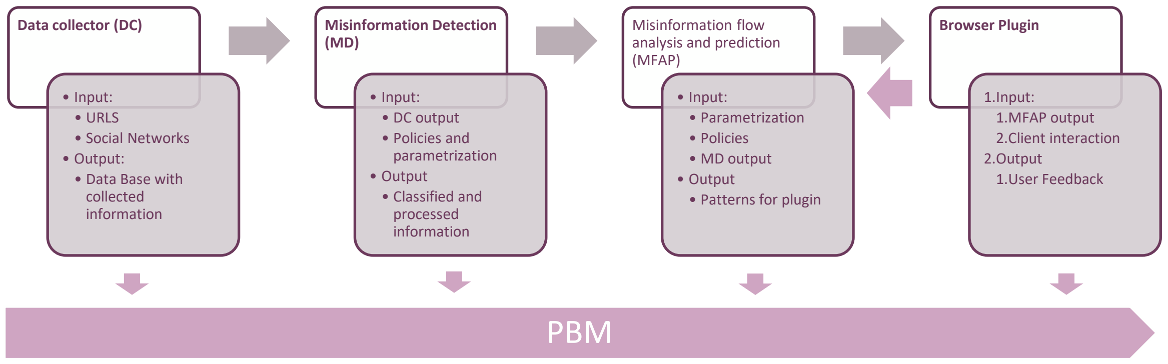

Figure 9 Data flow

3. Conclusions

This deliverable describes the high-level architecture of the Co-Inform platform, including the interaction between its expected components. Specifically, it provides a high-level specification for the platform components and their expected interaction, namely:

- Data Collector (DC)
- Misinformation Detection (MD)
- Misinformation Flow Analysis and Prediction (MFAP)
- Browser Plugin
- Perceptions and Behavior Mining (PBM)

This is still a preliminary version of the architecture, which will be revised, updated and detailed as required in order to comply with the set of requirements gathered with the Platform's end-users in the course of the activities within WP1, with the management policies from WP2 and with the services developed in WP3. Specifically, it will be updated in two additional iterations (D4.3, and D4.4), to be submitted respectively in M15, and M21.

Specifically, future versions of this deliverable should also specify:

- Specification of components
- Workflow and integration of components
- API documentation
- Hosting architecture