Co-inform

Context Matters,
Your Sources Too

Content Collection Services

D3.1

#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:	3.1 Content Collection Services		
Version:	1.0		
Work Package:	WP3		
Submission date:	29/03/2019		
Nature:	DEM+R	**Dissemination Level:**	Public
Lead Beneficiary:	OU		
Author(s):	Ronald Denaux – Senior Researcher, ESI Olga Salas – Research Engineer, ESI		
Contributions from:	SCYTL, OU, SU		

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	20/02/2019	Ronald Denaux	Table of Content
0.2	11/03/2019	Ronald Denaux	1st Draft
0.3	14/03/2019	Ronald Denaux	2nd Draft Ready for Internal Review
0.4	27/03/2019	Ronald Denaux	With Internal Review changes
1.0	29/03/2019	Ronald Denaux	Ready for submission

Disclaimer

Copyright Message

Executive Summary

This document describes the Co-Inform Content Collection Services, one of the main technical components in Co-Inform which is responsible for collecting public content from the web that is relevant to the various Co-Inform pilots and to fact-checking.

The collected content needs to be processed in order to strip away non-relevant markup and the raw text needs to be enriched with semantically relevant information such as categories (topics the content talks about such as “social issues”, “immigration”, “housing”, etc.), entities (people, places, organizations, etc.) and claims (assertions being made that can be fact-checked). The collected and enriched content then needs to be stored in a way that facilitates finding content using a variety of search and filtering options. Such search capabilities will be used by other technical Co-Inform partners to build high-level misinformation detection, analysis and prediction services.

This document first describes the requirements for the Content Collection services in a non-technical manner. Next, it describes how the services have been implemented at a conceptual level, which requires some technical knowledge, but should be understandable to non-technical audiences as well. Finally, this document describes the Application Programmer Interface (API) of the services, which is how technical partners will be able to access the services; the API is described by providing examples of various search capabilities.

Table of Contents

Table of Figures

Glossary Table

Term	Definition
API	Application Programmer Interface, defines how programmers can use a service or library
JSON	JavaScript Object Notation
REST	Representational State Transfer, an architectural style for offering web-based service interfaces
RSS	Really Simple Syndication (or RDF Site Summary)

1. Introduction

This document describes the content collection services developed in T3.1 of the Co-Inform project.

The main objective of these services, according to the Co-Inform Grant Agreement is to "collect content from the Web, including news sources, public social media content, blogs, and fact checking websites. The content will be processed and semantically structured, enriched, and integrated, to provide a networked conceptual representation of this content, ready for T3.2 and T3.3 analysis.".

One of the main challenges when developing such a service in the context of Co-Inform is that it needs to be flexible enough to allow for the co-creation paradigm used within the project. That is, the solution needs to be able to adapt to new requirements as users demand these. At the same time, we need to provide a service that is already useful during the first stages of the project to support the development of disinformation analysis services. Also, the produced services need to be robust and scalable to be able to handle large volumes of content.

In order to meet these challenges, we have taken into account early requirements discussed during the first few Co-Inform meetings. Also, we have reused existing solutions for content collection and analysis developed at Expert System.

This document is structured as follows: in Section 2 we describe the requirements for the Co-Inform Content Collection services; this is done at high-level of abstraction that can be understood without much technical background. In Section 3 we describe how we have implemented our services and provide an overall architecture and describing the implemented capabilities in detail; this section gets into more technical detail but is generally written at a conceptual level and provides various examples to make it understandable for non-technical readers. In Section 4 we describe the search API, which is how other technical partners will use our services; this section is aimed at a technical audience and provides various examples of how to find documents based on search capabilities supported by our services. Finally, in Section 5 we draw conclusions and discuss possible improvements based on current limitations of the services.

2. Service Requirements

In this section we discuss the main capabilities and limitations that need to be provided by the content collection services. Besides listing the main requirements, this section also serves to introduce and define some technical terminology.

2.1. Language Coverage

Within the Co-Inform project, pilots will be performed in three different locations: Sweden, Greece and Austria. All these pilots will include local people; hence it is important for the content collection services to be able to process texts in the local languages. The languages to be supported are:

- **English** (as the main language used to collaborate between partners in the project and to disseminate the results)
- **Swedish**, to support the Swedish pilot
- **Greek**, to support the Greek pilot
- **German**, to support the Austrian pilot

Optionally, other languages may be supported such as Spanish.

2.2. Content Sources

Content Sources refer to sources of text data. As stated in the Grant Agreement, in Co-Inform we rely on content that has been published on the Web and that is publicly accessible. With other words, we explicitly exclude the so called Dark-Web (which is content on the web, but that requires special access in the form of user names and passwords). Furthermore, the Web as such is too large to include as a single "source", instead, we have to define more specific sources. The Grant Agreement specifies various types of content sources: "news sources, public social media content, blogs and fact checking sites". On a more technical level, we need more specific protocols to define sources. In particular, we see the need for the following sources:

- **Web feeds**: are data formats that enable syndication, i.e. allows publishers to notify subscribers about content updates. Most websites now support some form of feed to notify about new content, like new articles or blogposts. The two most common web feeds are RSS[1] and Atom[2]. Most news sites provide multiple feeds to subscribe

[1] http://www.whatisrss.com/

[2] https://validator.w3.org/feed/docs/atom.html#whatIsAtom

to specific categories of news. For example, The Guardian provides RSS feeds to specific subtopics, such as US immigration[3], based on their internal categorization. Also, most blogs are built on top of frameworks which provide support for web feeds. Finally, most fact-checking websites, like news sites, also provide web feeds.

- **Public social media content**: the two main social media sites to support are Twitter and Facebook. Unfortunately, neither provides RSS feeds hence these sources need to be handled separately. Both social media networks provide custom APIs to interact with their data and require setting up developer accounts. Both APIs tend to be limited in the number of requests that can be made per day/hour. In both cases the APIs provide ways to gather content based on users on the respective network. It should also be possible to gather content based on hashtags. In summary, supporting content collection from these social media sites requires custom development. Although Twitter and Facebook should be supported, other social media sites besides these two can be considered in the future; but note that they will also have the same type of restrictions.

- **Keyword-based sources**: Although most websites provide web feeds, there are some which do not. Also, some web feeds may be too general, resulting in too many daily documents which are not relevant to Co-Inform. For this reason, it may be more appropriate to define content sources based on one or more keywords. This works by using a search-engine and querying for the defined keywords on a regular interval (e.g. once a day). The search results can be from a pre-defined list of sites and can be included as relevant content.

In summary, the content collection service **should be able to crawl web content using (i) web feeds (in particular RSS) as well as (ii) keyword-based sources (via a search engine) and (iii) public social media content from Twitter and Facebook (using their respective APIs)**.

i Content Metadata

Since the collected content will be used by other Co-Inform modules to perform high-level analysis, it is important to keep track of metadata to help to identify the content and its publisher. The collected content should, whenever possible, contain metadata about:

- The **source**, to understand where the content came from:
    - The type of source: i.e. webcrawl, social media, keyword-based.

[3] https://www.theguardian.com/us-news/usimmigration/rss

- A source id: this is an identifier for the publisher. E.g. all the content published by Snopes can be assigned an id "snopes.com".
- The URL: the public address for the content

- **Relevant dates**, to understand how the content may have changed and when the content became available. Since different publishers may report time using different formats, the content collector should also perform some normalization of the dates to make it easier to query, aggregate and filter by dates. In particular:
  - The publishing date: when the content was originally published (this is typically reported by publisher)
  - A last modified date: when the content was last modified. Typically, content may need to be modified after the original publication date. This date, in combination with the publishing date can be useful to detect changes.
  - Collection date: when the content was collected by the content collection service.

- The **main textual content** that needs to be analyzed. In particular:
  - The title: as assigned by the publisher
  - The textual content: as extracted by the content collector. Web content is usually published within HTML markup; however, for text processing the HTML markup needs to be stripped away to obtain a pure text version.
  - A digest: for the textual content. This is a relatively short code that can be quickly computed and can be used to check whether content has changed. A digest is also useful for detecting duplicate content: e.g. whether the same article was published at two different websites.
  - The size: this can be used to filter content by the textual size. Some content can be very short (e.g. tweets) while other can be very long (e.g. long reads).

2.3. Semantic Enrichment

Semantic Enrichment is the process of automatic analysis of textual content. Within Co-Inform this process corresponds to what was promised in the Grant Agreement, that the content "will processed and semantically structured, enriched, and integrated, to provide a networked conceptual representation of this content, ready for T3.2 and T3.3 analysis". In practice, this enrichment means automatically categorizing texts, extracting (named) entities and extracting claims and claim reviews. In the following subsections, we describe each of these subtasks in more detail.

i Categories

One common way to enrich raw textual content is to categorize it. This amounts to associating the raw content to a pre-defined set of "categories".

Once content has been categorized, users can filter and select only content that is related to categories relevant to a task at hand. Another use of categories is that it facilitates finding similar documents (since they will have similar categories). A final use of categories is analysis of trends: for example, you may be able to see that a particular news source publishes mainly content about a small subset of categories.

Since we do not know in advance what the topics of fake news may be, this service **should support a wide set of generic categories by default**. As the project progresses, we **may need to focus on a more fine-grained set of categories** (e.g. on housing, immigration or refugees). Relevant to this requirement is that the set of **categories should be well documented, easy to inspect and understand** and **categories should be organized in some intuitive way**. For example, it is common practice to organize categories and subcategories in tree-like structured called taxonomies.

ii (Named) Entities

Another common way to enrich raw textual content is via "Named Entity Recognition". This is an NLP task whereby mentions of real-world entities (i.e. people, places, organizations) are extracted automatically. This allows users to filter content by entities of interest (e.g. see only content that mentions a specific person and/or place). Like categorization, named entities, facilitate content comparison and analysis of trends across large collections of content.

The most frequent types of entities are people, places and organizations, but other types of entities are also useful: dates, time references, buildings, chemicals, etc. In particular, having special subtypes of entities can be useful; e.g. having subtype "politician" (a type of person) can be useful, instead of having to search for all known politicians.

For Co-Inform, the content collection service **should extract the main named entities: People, Places and Organizations**. Optionally, the content collection service **may extract more specific types of (named) entities**.

iii Claim (Review) Extraction

Categories are useful enrichments because they provide a summary at the document level and entities are useful because they provide an overview of real-world entities mentioned in the content. However, neither of these enrichments is directly useful for assessing the veracity of the content. The reason for this is that veracity is a quality of a claim or statement, which typically occurs at the sentence level. For this reason, the content collection service **should be able to identify, and extract claims** from the raw textual content.

As opposed to categorization and named entity recognition, claim extraction is a relatively novel and challenging task in NLP. The main challenge here is to be able to distinguish between a claim and other types of sentences. In particular, it can be difficult to humans, let alone automated systems, to distinguish between an opinion and a claim. While categorization and named entity recognition tasks typically achieve around 80 to 90% accuracy in tests, such high results should not be expected from claim detection systems yet. Claim detection is considered a subtask in fact-checking and is useful on its own right: it can be useful for fact-checkers to select which claims to check. Also, it can be useful to analyze trends in claims and to compare claims.

Although it may be useful, for the content collection service in Co-Inform, we initially limit ourselves to detection of claims, but we do not aim to extract additional metadata about the claim like detecting the author of the claim or categorizing the claim (both of these may be achievable using the collected content but are not requirements).

Related to claim detection is the task of a **claim review extraction**: i.e. not only detecting a claim, but also detecting that the content is reviewing the claim: i.e. assessing its veracity and assigning it some kind of *veracity score*. Automated extraction of such claim reviews is not currently possible and is an active area of research. However, many fact-checking sites are publishing metadata using a common data vocabulary called ClaimReview[4]. This is part of schema.org, a shared vocabulary proposed by the main web search engine providers (Google, Microsoft, Yahoo, Yandex), which allows web content providers to publish structured data as part of their websites. This allows such sites to appear higher on search results and it allows search engines to provide more accurate search results. Since such claim reviews are highly relevant to Co-Inform, the content collection service **should support extracting schema.org ClaimReview structured data**.

[4] https://schema.org/ClaimReview

2.4. Storage, Indexing and Search

The content collector should provide a means for other Co-Inform partners to search the collection of crawled and analyzed content. In particular, partners involved in T3.2 and T3.3 should be able to search the content in order to perform misinformation detection, flow detection and prediction. In technical terms this means that the content collector **should store the crawled content, along with its metadata and semantic enrichments**.

The **stored content should be indexed**, this is a process to organize the stored data in such a way that it is easy to retrieve using special entry points to the data (e.g. a date index makes it possible to search by a specific date or a date range). Finally, the content collector **should provide a search API**, an interface that programmers can use to retrieve and analyze documents. To facilitate tasks T3.2 and T3.3, this search API should support:

- **Keyword search**: that is finding content that matches one or more keywords
- **Content metadata search**: for finding content based on publisher, published date, etc.
- **Semantic enrichment search**: for finding content by categories, (named) entities, claim and/or claim reviews
- **Paging**: for iterating through search results in an efficient manner. This is especially relevant when a large number of documents match the query. By default, only a small number of results should be returned, but the user can request subsequent “pages” in order to retrieve all of the results.
- **Semantic enrichment facets**: for providing an overview of the main semantic enrichment categories and entities that are relevant for a search. This is also useful for searches that return a large number of results as it enables the user to see options for refining the search. For example, if a search has 1,000 results, a category facet will show the 5 most frequent categories for the search results, stating that 800 of the results have category X and 600 of the results have category Y.

Naturally, the user should be able to combine these different types of queries to produce intricate queries. For example, the API should allow a user to express the query “find documents published by The Guardian about immigration and housing, published in 2017 and mentioning Boris Johnson.”. Just to be clear, the content collector will not provide a natural language interface for searching: the user will either need to know how to translate this query to the format specified by the search API, or the user will utilise an appropriate graphical user interface which composes the query based on, for example, a form.

3. Content Collection Server and Database

As part of T3.1 we have set up a content collection server and database that meets the capabilities described in Section 2. Here, we describe our implementation to provide an idea of how it works at a conceptual level. Although having a conceptual understanding of how the content collector works is useful, note that knowledge about many of the implementation details are not strictly required for Co-Inform partners, who will directly interact with the service via the REST APIs described in Section 4.

3.1. Architectural Overview

Figure 1 shows the overall architecture of the Co-Inform Content Collector. In this section we shortly describe each of the components, but most of them are described in more detail in subsequent sections.

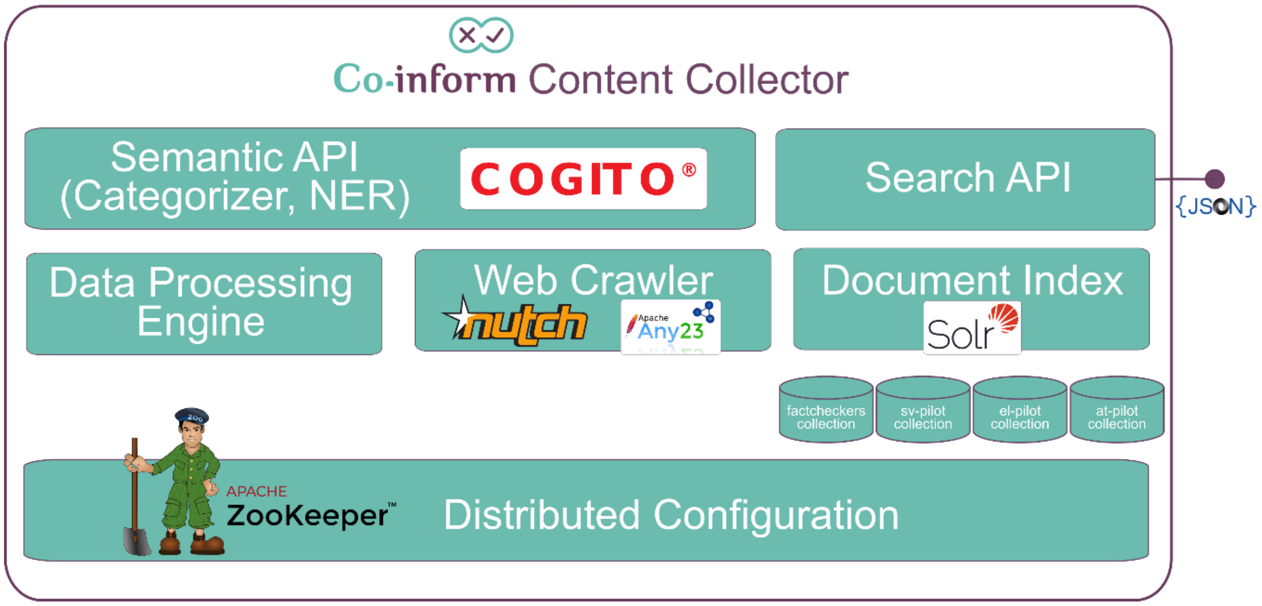

Figure 1 Architecture of the Content Collector

As the figure shows, the Content Collector is built on top of a **distributed configuration** layer, implemented using Apache Zookeeper. This layer ensures that the overall service is scalable because all the components can be configured and deployed in a distributed manner. For example, one deployment can be to run all the components in a single server, but we can also choose to use a cloud-based deployment where there are several servers running each component.

The core of the Content Collector is the **Data Processing Engine**, which maintains a set of *sources* and target *document collections*. The data processing engine acts as the orchestrator and triggers new web crawls based on a schedule configuration and monitors when new crawled documents are added to their respective collections in the document index. When needed it triggers automatic machine translation of the raw texts and triggers semantic enrichment of the texts.

The **Web Crawler** is responsible for crawling websites, retrieving the HTML content and filtering the raw textual content that can be analysed. The crawler also is responsible for extracting metadata that is related to the crawled website. We have also added a plugin that also extracts structured data published as part of the website, in particular schema.org ClaimReviews, which are the main results of fact-checkers. The Web Crawler is implemented using Apache Nutch, and the structured data and ClaimReview extractors are based on the Apache Any23 project. More details about this component is provided below.

The **Document Index** is software that is able to create and manage various document databases, called collections. These collections are stored in such a way that they can be searched quickly. We use Apache Solr as the underlying implementation for the document index. Within Co-Inform we maintain 4 collections, one for each of the pilots (Sweden, Greece and Austria) and a generic one for indexing fact-checking sites. Each of these collections is populated through a list of relevant sources (the various implemented source types are described below) for the specific pilot. For example, for the Swedish pilot, we have defined sources for the most used news websites in Sweden as well as various smaller (local and topic focused) websites and blogs as well as publicly available social media posts based on keywords, hashtags and user accounts that are relevant to the pilot. Besides the 4 main Co-Inform collections, custom collections can be added if necessary.

The **Semantic API** is responsible for performing semantic enrichment of the text contents. This component is based on Expert System's Cogito technology and described below in further detail.

Finally, the overall Content Collection service is accessible via a RESTful **Search API**, this means that programmers can access the service via commonly used web protocols (HTTP and HTTPS) and process the available information using the JSON data exchange format. This again is very common and makes it easy to search the collected content. The JSON format also makes it very easy to build web and mobile application interfaces on top of this service that end users can utilize. Section 4 describes this interface in detail.

3.2. Content Source Types

Our implementation meets the requirements for content source types as described in Section 2. More specifically, we support RSS feeds, Twitter sources, and keyword-based sources. We describe these in more detail below.

For each of the source types, the content collector can be configured to specify how often the source should be crawled. By default, we crawl sources once a week, but depending on the source, we can choose to crawl every hour (e.g. when following a trending topic on Twitter), or only once (e.g. when crawling a set of archived fact-checking articles that are no longer being updated).

i RSS feeds

We support defining sources based on one or more RSS feed URLs. Each source can be given a unique name, that will be shown as part of the collected content metadata. For example, we can define a "Guardian housing and immigration" source, which points to various RSS feed URLs for US and UK immigration and housing subcategories. This allows for fine-grained definition of sources.

We also support exporting and importing these lists of RSS feeds using the OPML (Outline Processor Markup Language), a data format commonly used by web feed aggregator applications[5].

ii Twitter API Sources

We support defining sources based on the Twitter developer API. We support the basic standard (i.e. non-premium) operators[6], which include defining source per keywords, phrase, disjunction (OR), exclusion, hashtag, account, dates, replies to an account, Twitter-provided filters (e.g. potentially sensitive content, image or video).

As with RSS feeds, we can associate multiple Twitter keywords to a single Twitter source for added flexibility (e.g. to search for keywords in different languages), or to modify the queries without needing to define new sources.

iii Facebook API Sources

We support defining sources based on a Facebook app that is created through a Facebook Developer Account. This app can be used to crawl posts and comments from public pages and groups. These sources are rate-limited by Facebook, which means we are only allowed

[5] https://en.wikipedia.org/wiki/OPML
[6] https://developer.twitter.com/en/docs/tweets/rules-and-filtering/overview/standard-operators

to request content from Facebook a certain amount of times per hour/day. This means that in practice, we will need to select Facebook sources sparingly.

iv Keyword-based Sources

We support defining sources based on keywords, or to be more precise: on *search engine queries*. We have implemented support for the main search engines: Google, Bing and/or Yahoo. By default, every time this source is crawled, we keep the first 50 results returned by the selected search engine(s).

Because we work at the level of queries, we can use the advanced query operators provided by the different search engines to define fined-grained sources. For example, we can use the site operator to only return results for a specific website (presumably one that does not support RSS feeds, or only offers a very broad feed), match exact phrases using double quotes and exclude words using the minus (-) character. Each search engine has its own set of advanced search operators; thus we refer to the documentation of Google[7], Bing[8] and Yahoo[9].

v Nutch Web Crawl Sources

Since our implementation is based on Apache Nutch, it is also possible to create a source based on a Web Crawl. In this case, the source is defined by one or more "seed" URLs which are used as starting points. The crawler then adds those websites, but also collects URLs which are linked from those seed pages and collects those as well in the following iteration. Therefore, besides the seed URLs, this type of sources also requires the definition of a maximum iteration. Due to the exponential nature of this type of source, we typically recommend a low value for the maximum iteration (e.g. 2).

3.3. Machine Translation

In order to support the required languages, the Content Collector we have implemented uses a machine translation service provided by Systran[10]. In this section, we motivate the need for using machine translation and discuss the advantages and disadvantages of this approach.

In short, the main motivation for using ML is that providing native semantic enrichment for each individual language is prohibitively expensive. To understand why, we need to explain

[7] https://support.google.com/websearch/answer/2466433?hl=en
[8] https://docs.microsoft.com/en-us/previous-versions/bing/search/ff795620(v=msdn.10)
[9] https://search.yahoo.com//web/advanced
[10] http://www.systransoft.com/

how the Semantic Enrichment performed by our Content Collector works. The semantic enrichment is based on Expert System's Cogito technology. Although Cogito provides native support for 14 languages, not all of these 14 languages have the same level of support. At its core, Cogito performs the usual NLP tasks of tokenization, part-of-speech detection, lemmatization and sentence splitting. Then Cogito performs word sense disambiguation. After that, rules are executed to extract (named) entities and to perform categorization (the results of which are what is typically understood as enrichment). The main issue is that "native support" is considered achieved when Cogito is able to perform word sense disambiguation; however the most mature languages (English, Italian, Spanish) each have over 100K manually crafted rules to perform NER and Categorization for our main taxonomies (Intelligence, Crime, Cybercrime, Terrorism, Geography and Emotions), which are needed to achieve a high level of precision and recall. Developing such rules for other languages requires person-years and is an effort that is out of the scope of Co-Inform.

By using ML in this project, we can use English as our core language for Semantic Enrichment, and we will use a Machine Translation service to translate from Swedish, Greek and German to English. This is a common solution that works well in practice. Although the results are not as good as they would be if we had mature native support for these languages, the quality of machine translation is currently good enough to be useful. In general, while accuracy for English texts is around 90%, accuracy for translated texts can be in the range of 80 to 85%, depending on the quality of the translation and the quality of the text itself. Hence, this solution will be sufficient to asses the value of the Co-Inform approach. Switching to a full native support for different language can be done after the project as part of the exploitation of the project results if the extra 5 to 10% in accuracy is desired.

One advantage of using a core language is that all texts can be inspected by English speakers, even if they do not speak the original language of the content (Swedish, Greek or German), making it easier to verify that technical services are working as intended.

3.4. Schema.org ClaimReview extraction

In order to support ClaimReview extraction, we have implemented an extension for the any23 Apache Nutch plugin, that focuses on this task. Whenever the Content Collector crawls a web document from one of the defined sources, it extracts any structured data included in the web document. Then, if the document contains schema.org ClaimReview items, it extracts the main information about those items to make them easier to find once they are stored in the document database.

Although ClaimReview is a vocabulary for structured data, it is not overly restrictive and leaves room for the data publisher to publish the data in a way that is comfortable for them. This means that there is some variance to the ClaimReview items that are found on the web. Our crawler does not perform unification of the values, it simply extracts some of the main attributes of ClaimReviews. In particular it extracts, when available:

- The claimReviewed: this is a string value. Some fact-checkers phrase the claims as questions (e.g. Are 60% against “The Backstop” in Northern Ireland?) while others phrase them as assertions (60% in Northern Ireland are against “The Backstop”).
- The author name: a string identifying the fact-checker, e.g. “Snopes”
- The rating altName: a string summarizing the result of the fact-check. Each fact-checker uses their own set of rankings, hence these values need to be normalized.

Our plugin also extracts all the structured metadata related to each ClaimReview and stores it in the n3 format. This is essentially a subset of all the structured metadata available and can be useful in cases where the triples contain additional information that is not captured by the extracted main attributes.

We intend to publish the resulting Nutch plugin as an open source project on GitHub.

Finally, although an increasingly large percentage of the fact-checkers publish schema.org ClaimReview data, not all of them do. In particular, some high-quality disinformation sites like “EU vs Disinfo”[11] may publish relevant fact-checking articles without providing structured data. We may consider building additional custom scrapers (Nutch plugins) for such websites as part of Co-Inform if we see a need for this.

3.5. Semantic Enrichment via Cogito

Expert System’s semantic text analysis technology is called Cogito. At its core, Cogito performs the usual NLP tasks of tokenization, part-of-speech detection, lemmatization and sentence splitting. Then Cogito performs word sense disambiguation. After that, rules are executed to extract (named) entities and to perform categorization and claim extraction.

i Categories

For categorization, Cogito provides 6 standard taxonomies:

- **Intelligence**: this is a generic taxonomy that contains categories relevant to the intelligence and security domain. Since many areas of disinformation are related to these categories, being able to filter by these will be useful. This taxonomy is

[11] https://euvsdisinfo.eu/

organized in up to 4 levels of subcategories. Some of the broad categories are “Arts, Culture and Entertainment” and “Environmental Issue”, while some of the more narrow categories include “Air Pollution”, “Genetics” and “Housing and Urban Planning”.

- **Crime**: this is a taxonomy focused on crimes. This includes broad categories such as “Property crime” and “Positive results from investigations and measures”; and narrow categories such as “Immigration-related offenses”, “Acquittals”
- **Cybercrime**: this is a taxonomy focused on illegal activities online with broad categories such as “Cyber Attacks” and “Cyber Security” to narrow categories such as “Zero-day”, “Ransomware” and “Defacing”.
- **Terrorism**: is a taxonomy focused on issues related to terrorism. In particular, it provides different ways to categorizing content via broad categories like “Terrorism by matrix”, which has subcategories “Religiously inspired terrorism” and “Narco-terrorism”. Other broad categories include “Terrorist activities and tactics” and “Counterterrorism”.
- **Emotions**: this is a relatively flat taxonomy (i.e. it is not nested into many layers of subcategories) that describes many emotional states such as “Joy”, “Hope”, “Forgiveness”, “Confusion”, “Regret”, “Repulsion” and “Materialism”.
- **Geo**: provides categories for each country (and for the US for each state)

As part of the technical documentation we provide to partners in Co-Inform, we include XML files listing all of the categories in each of these taxonomies.

ii (Named) Entities

For (named) entity extraction, Cogito provides the standard named entities: People, Organizations and Places. However, Cogito leverages additional information about known entities to provide more specific types with the additional advantage that in this case we can normalize the names. For example, Facebook can appear as “Facebook Inc”, or simply as “Facebook”.

- People: Note that the subtypes can be overlapping either per definition (US Senators are also World Leaders and are also Influential People) or because the person in question fulfills different roles (e.g. people who have both a leading role in companies as political functions)
    - **US Senators**: e.g. “Kamala D. Harris” or “Ted Cruz”
    - **Key Company Executives**: e.g. “Tim Cook” or “Michael O’Leary”
    - **World Leaders**: typically, leaders of nations and large organizations
    - **Influential People**: any other types of famous people
    - **Wanted for Terrorism**: e.g. “Abu Bakr al-Baghdadi”

  - The **default** value can contain all of the “known people” from the categories described above, as well as “unknown” people. Values can be e.g. “Speaker” or “Johnson”.
- Organizations:
  - **Terrorist Organizations** e.g. “ISIS” or “al Shabaab”
  - **Public Leading Companies** e.g. “Facebook Inc”, “Airbus”
  - **Non Governative Organizations**: e.g. “Oxfam International” or “Human Rights Watch”
  - **Intelligence Agencies**: e.g. “Federal Bureau of Investigation” or “Europol”
  - **Criminal Organizations**: e.g. “Mara Salvatrucha” or “Aryan Brotherhood”
  - The **default** value can contain any of the above as well as unknown organizations extracted based on their context in the text.
- Places:
  - **Points Of Interest**: can contain well-known buildings or places e.g. “White House”, “Royal Institution”, “V&A” or “Circus Maximus”.
  - **Military Facilities**: e.g. “Dover AFB” or “Naval Observatory”
  - **Critical Infrastructures**: e.g. “hospital”, “airport”, “harbour”, “bus”, “train”, “terminal” or “water company”.
  - **Buildings** e.g. “hospital”, “parliament”, “hotel”, “church”.
  - **Airports**: e.g. “Heathrow Airport”, “Gatwick”, “Arlanda”
  - The **default** value can contain names of countries, cities, towns, states, provinces, but also e.g. “Atlantic Ocean”. When the place is known we also can provide geographic coordinates to e.g. locate the place on a world map.

Besides these entities we also provide various specialized entities:

- **Weapons of Mass Destruction**: e.g. “chemical weapon”, “cruise missile”, but also “novichok” and “mercury”.
- **Vehicles Motorbikes** e.g. “Harley Davidson”, “Lime-S”
- **Vehicles Cars**: e.g. “electric car”, “Ford”, “Toyota”
- **Natural Disasters** e.g. “flood”, “storm”, “plague”
- **Military Forces** e.g. “army”, “militia”, “land forces”
- **Military Equipment** e.g. “gun”, “ammunition”, “F22”
- **Military Actions**: e.g. “battle”, “mission”, “capture”
- **Medical Treatments** e.g. “palliative care”, “immunotherapy”
- **Medical Conditions** e.g. “infection”, “diabetes”
- **Illicit Drugs**: e.g. “heroin”, but also slang that can refer to such drugs like “snow”
- **EW Devices** e.g. “radar”, “access point”, “defense system”
- **Chemical Agents** e.g. “tear gas”, “chlorine”, “nicotine”
- **Biological Agents** e.g. “salmonella”, “Ebola”
- **Hash** e.g. “B5F6EDCDF90DCC10286FC0BC599D94CAB8735DFB”

- **Vulnerabilities** e.g. “glitch”, “loophole”, “false negative”
- **Security Tools** e.g. “data loss prevention”
- **IP** internet protocol addresses e.g. “10.194.118.106”
- **Malware** e.g. “rogue software”, “virus”, “spyware”
- **Cyber Illegal** e.g. “troll”, “cookie”, “piracy”
- **Cyber Attacks** e.g. “hacking”, “HTML injection”
- **Social Tags** e.g. “@PolitiFact”, “#Brexit”

As described below, all of these entity types can be used to filter and to refine searches.

iii Claim Extraction

Because the semantic enrichment, in particular the categorization, is based on rules, we can use a subset of these rules to extract phrases in the text that have a high probability of being “claims”. Here, we refer to claims as assertions of something that can be verified. As discussed above, the task of claim extraction is still actively being researched; hence it is currently not possible to distinguish accurately between a claim and an opinion.

Because we extract claims based on the handcrafted categorization rules, we can also categorize the claims based on the same 6 taxonomies described above. We select a couple of example claims which are illustrative for both the good and bad cases which we are capable of extracting:

- “In England, the elderly in particular voted for Brexit”. Extracted from a German article on welt.de[12]. Categorized as “politics” from the Intelligence taxonomy. In this example, the claim is apt for fact checking.
- “Of course, no mosque is taking in Christian refugees, who are the prime targets of devout Muslims who understand that when Allah commanded them to slaughter the ‘people of the book,’ he meant it literally”. Extracted from a site well-known for spreading disinformation[13]. Categorized as “Offences against public safety” in the Crime taxonomy as well as “terrorist activities and tactics” in the Terrorism taxonomy. In this case the extracted phrase is not a clear assertion, but rather a combination of various claims, some of which are hard or impossible to verify.

[12] https://www.welt.de/wirtschaft/bilanz/
[13] https://madworldnews.com/liberal-church-muslim-refugees/

3.6. Content Storage and Indexing

After crawling, the extracted text and metadata (including structured data and ClaimReviews if available) are stored in a Solr database, more specifically in a collection in Solr. In parallel, the Data Processing Engine triggers a possible machine translation as well as the semantic analysis. When the semantic enrichment process is finished, the document in the Solr is updated by adding the semantic enrichment fields.

The Solr instance is configured in such a way that it is possible to query for documents based on keywords in the textual fields or by looking up substrings in the various semantic enrichments.

We also exploit the Solr functionality of **facet fields**, which provides a way to provide summaries of how many documents in the search result have specific values for specific fields. For example, suppose we have a search that matches 10,000 documents. Then, we can use the categories field as a facet field, so that the result will be (besides the first 10 documents), an overview of the facets, stating that 60,000 of the matching documents have a category from the Crime taxonomy, 80,000 a category from the Intelligence taxonomy, etc. This is typically useful to enable interactive refinement of the search, whereby a use 'selects' a specific subcategory to make their search more specific.

Since each of the Co-Inform pilots will have different stakeholders and potentially different topics, we have chosen to use different collections of documents for each pilot.

3.7. REST API

The final component is the REST API. This component provides a programmatic interface for other Co-Inform partners to search for documents that have been collected. The REST interface emulates the default Solr REST API for searching through a collection, but this is not a simple redirection.

One of the tasks of this component is to provide a more intuitive interface for Co-Inform. The main reason for this is that the Solr documents contains various private fields which are not relevant for Co-Inform partners. Also, the Solr documents often contain code values which are not human readable; for example, we store internal code identifiers for the categories in the various taxonomies. To make the Co-Inform REST API easier to use, this component automatically translates these internal category codes into human-readable versions.

Another important task of this component is authentication. Solr provides some basic authentication, but this can be difficult to configure and adapt. For this reason, this component implements authentication in a manner that should be easier for us to maintain. This means that, if we choose, we can require users of this REST API to provide authentication credentials (e.g. username and password). We also have some initial support for rate limiting to avoid having certain users making too many consecutive requests and overloading the server. This is in general good practice and makes the overall Content Collection Service more robust and ready for production.

The following section describes the REST API in more detail.

4. API

We expose the contents crawled and semantically enriched via a RESTful API which accepts a query string and returns JSON documents. In this report, we describe the main use cases available using this API. A full specification is described both as a Swagger specification and on an internal API documentation wiki (available to other Co-Inform partners). Naturally, as the services are used in practice, we are open to adapt and improve the API to support the implementation of further Co-Inform tasks.

4.1. Design Decisions

Most of these have been described above, but we repeat them here in a concise manner.

Whenever you search for documents *you must specify a collection*, which are there to keep documents which are relevant to one Co-Inform pilot separate from that of other pilots. We provide an endpoint to get an overview of the available collections.

We have chosen to provide a *REST interface*, since this is the most common way to provide web services. In the future we may consider providing a GraphQL interface if there is sufficient demand and a technical reason for it.

We have chosen to *only support JSON* to focus our efforts. Other serialization formats, such as XML can be provided in the future if needed.

4.2. collections Service

Method: GET

Endpoint: api/v1/collections

Parameters: None

This endpoint returns a map with a list of collections. Each collection is represented as a JSON object with fields description and name. The “name” is the value that you must pass to the search endpoint.

Example result:

```
{
  "collections": [
    {
      "description": "Collection for fact-checking issues",
      "name": "co-inform"
    },
    {
      "description": "Terrorist and isis information analysis",
      "name": "dante"
    },
    {
      "description": "test collection",
      "name": "value"
    },
    {
      "description": "asdf",
      "name": "asdf"
    }
  ]
}
```

4.3. Content search Service

The search endpoint is the main endpoint provided by this API.

Method: GET

Endpoint: api/v1/search

There are a variety of parameters you can provide to specify your search. For this reason, we first discuss generic parameters and then discuss parameters needed for specific use-cases below. The main generic parameters are:

- collection: the value should be one of the collection names returned by the collection's endpoint.
- q for performing a basic keyword search over all the content fields in documents.
- q_<document_field_name> for performing search over specific fields in the documents. This requires that you know the field names that documents can have. Below we will see several examples.

i Response Data Structures

A successful response to the search endpoint will be a JSON object with main fields:

- responseHeader: provides an overview of the parameters which were recognized and a time in milliseconds that was needed to produce the response
- response: this is the main part of the response and is an object with subfields:
  - numFound: an integer with the total number of documents matched (not all of which are included in the docs)
  - start: the start index of the page being returned
  - docs: a list of document objects representing a "page" of documents (see below for how to control the page size). By default, we return 10 documents per page. Each document is a JSON object with a large number of fields for the textual content, the document metadata and semantic enrichments. The

full list of possible document fields is provided as part of the Swagger specification.

- facet_counts an optional object with the faceting information (see the subsection on faceting below).

Each document can contain a large number of fields, most of which are optional. Here, we provide an overview of the fields categorized by their source in the Content Collector pipeline:

- textual content:
  - content: the extracted content if originally in English, or the English translation of the content when originally in some other language. Note that this content has undergone normalization processes to make it more amenable for the semantic enrichment process. Any highlighting offsets provided in semantic enrichment fields refer to positions in this field.
  - content_language: the original content if not in English.
  - title: the title of the crawled website. Usually extracted from declared metadata in the header. This may be the English translation of title_language if it is not English.
  - title_language: the original title of the crawled website. Only available if the original website is not in English.
- content metadata:
  - contentLength: number of characters in the content field.
  - collection: the name of the Co-Inform collection of documents.
  - digest: a unique hash calculated based on the content value.
  - id: an identifier for this document
  - lang: 2-letter ISO-3166-1[14] code for the language used for analyzing the text. Usually ‘en’.
  - lang_orig: 2-letter ISO-3166-1 code for the language detected in the original content.
  - lastModified: datetime when the content was last modified
  - last_update: datetime when the document was last updated in the database
  - publishedDate: datetime when the content was first published
  - size: size in bytes of the originally crawled content (before stripping away html tags and normalizing the text)
  - source: type of source e.g. 'WebCrawl', 'Twitter', 'RSS feed', etc.
  - source_id: typically, the publisher or domain name
  - status: internal status of this document in the Content Collectors, typically 'done', but can be 'running' if document is still being analyzed

[14] https://en.wikipedia.org/wiki/ISO_3166-1

  - url: web address of the site where the content was extracted from
- structured data and schema.org ClaimReview:
  - structured_data_n3: a list of all the n3 triples extracted from the crawled site.
  - schema_org_claimReviewed: a string containing the claim that was reviewed
  - schema_org_cr_n3: a list fo all the n3 triples related to the ClaimReview
  - schema_org_cr_author_name: a string with the declared author of the review
  - schema_org_cr_author_url: a URL where the author is described (e.g. a homepage)
  - schema_org_cr_url: the "canonical" url for the claim review
  - schema_org_cr_datePublished: the datetime when the claim review was published
  - schema_org_cr_rating_altName: the review result, a string assigned by the reviewer and not adhering to any common standard.
  - schema_org_cr_rating_value: a numerical value assigned to the rating result, relative to a scale between worst and best values
  - schema_org_cr_rating_worst: a numerical value defining the worst rating possible for a review
  - schema_org_cr_rating_best: a numerical value defining the best rating possible for a review.
- semantic enrichments: all of these are the results of the semantic enrichment process. All of these fields return by default the value followed by the offsets in the content where the category or entity was detected.
  - categories: a list of category paths e.g. "/Crime Taxonomy/Environmental crime/Pollution emissions and spills"
  - category_claims: a list of category paths followed by the offset pairs in the content where the claims have been identified.
  - a long list of fields for all the entities described in the section describing named entities: airports, biological_agents, buildings, chemical_agents, etc.

ii Default search

If you only specify a collection, but do not specify any filters (keywords, categories, etc.) to match, by default, we will return 10 documents in the collection.

iii Search by keyword

The most basic search is a search by keyword. This is controlled by parameter q.

Example queries include:

- search?collection=co-inform&q=immigrant

- search?collection=co-inform&q=muslim immigrant

This will search through the content field in all documents.

iv Multilingual search by keyword

The content field is always in English. In some cases, you may want to provide search in the original language of the content. For example, if your collection only contains sources in Swedish, you may want to search for a specific Swedish word or phrase. In such cases you can use a combination of parameters as follows:

search?collection=co-inform&q_content_orig=halvsanningar&q_lang_orig=sv

This exploits two specific fields in documents:

- lang_orig: this is the ISO two-letter code for the original language of the document. Relevant for Co-Inform are the codes sv for Swedish, de for German and he for Greek.
- content_orig: this is the original content as extracted during crawling (before applying machine translation). The content field always will contain English text.

The query shown, specifies that we want to find documents that were originally in Swedish and that contain the word “halvsanningar” (half-truths).

v Search by Categories

A frequent search refinement will involve narrowing a search by only selecting documents that match a specific category. We assume that you have knowledge about the various taxonomies, since you need to specify the full path of the category. In practice, users will have a user interface where they will be able to select relevant categories. An example search specifying a category is:

search?collection=co-inform&q_categories=/Intelligence Taxonomy/Social Issue

This refers to document field

- categories which contains all the category paths found in a document.

The query shown specifies that we want to find documents that match the “Social Issue” category in the Intelligence taxonomy. Since this category has subcategories, documents matching these subcategories are also included in the response.

Naturally, you can specify multiple categories to further refine your search. For example:

search?collection=co-inform&q_categories=/Intelligence Taxonomy/Social

Issue&q_categories=/Emotions Taxonomy/Success

vi Search by Entities

Another frequent type of search is by entity. As with categories, this type of search is easier if you know the available values, hence using the faceting information described below is recommended.

An example query may be to search for a specific type of building:

search?collection=co-inform&q_buildings=concentration camp

which will return all those documents in the "co-inform" collection for which the semantic enrichment process found mentions of "concentration camp" buildings.

Like the other filters, these can be combined. For example:

search?collection=co-inform&q_buildings=concentration camp&q_people=Adolf Hitler

Cogito will tend to do a good job of disambiguation mentions of "Hitler" to "Adolf Hitler" if there is sufficient evidence. However, in some cases you may want to match any person with that surname. You can use wildcards as follows:

search?collection=co-inform&q_buildings=concentration camp&q_people=*Hitler

vii Paging of results

Depending on your query and the collection, a query may have several thousands of matching documents. For performance reasons, by default we only return a maximum of 10 matching documents per query. This is called a page as described in the section describing the default search. You can choose a different page and even a different number of documents per page by using parameters start and rows.

For example, if the user has inspected the first 10 default results but wants to see documents 11 to 20. You can issue the following query:

search?collection=co-inform&start=10

Note that start is zero-based, so you need to specify 10 instead of 11.

If you want to return 50 results per page, you can request:

search?collection=co-inform&rows=50

We have a hard limit of a maximum of 100 rows for performance reasons.

viii Faceting results

Faceting is a common feature in modern search engines and allows the user to inspect frequent values for certain fields. Typically, graphical user interfaces will use faceting under the hood to provide menus where the user can select values to refine their search.

For example, a default query with facets for the categories field is:

search?collection=co-inform&facet.field=categories

this will return something like (abbreviated for clarity):

```
{"facet_counts": {
  "facet_fields": {
    "categories": {
      "/Crime Taxonomy": 61022,
      …,
      "/Crime Taxonomy/Illegal Construction": 328,
      …,
      "/Geo Taxonomy/Venezuela": 1080,
      …
    }
  }
 },
 "response": {
   "docs" = […],
   "numFound": 301346,
   "start": 0
 },
 "responseHeader": {…}
}
```

This says that the query returned 301,346 matching documents, and 61,022 of them had a category from the Crime taxonomy; 328 of them were tagged with the “Illegal Construction” category from the Crime taxonomy and 1,080 of them were tagged with the “Venezuela” category from the Geo taxonomy. Hence, if the user is interested in Venezuela, they can add a q_categories parameter with the value and the result of that query should have 1080 total matching documents.

Besides the categories field, you can also use any of the entity’s fields (buildings, people, places, etc.) as well as some of the metadata fields for faceting (source, source_id, status, lang and lang_orig).

ix Field highlighting

Parameter hl can be used to turn off highlight offsets for fields that support them. By default, this value is assumed to be true and this results in values such as

```
"main_elements":["illegal alien|158-171|528-531",
          "PolitiFact|53-62",
          "price|76-80",
          "raise the price|64-80"]
```

Where the field value is followed by one or more pairs of numbers indicating the offsets in the content field associated to this value.

If you specify query hl=false, the relevant fragment in the response will look as follows:

```
"main_elements":["illegal alien",
          "PolitiFact",
          "price",
          "raise the price"]
```

4.4. ClaimReview Search Service

i Search by claim reviewed

You can use the content collector to search for keywords in claims that have been reviewed. For this you can use the schema_org_claimReviewed field. For example:

search?collection=factcheckers&q_schema_org_claimReviewed=immigration

This will return matching documents. For example, one document may contain field:

```
"schema_org_claimReviewed":["Lack of affordable healthcare accounts for
a greater number of deaths per year than do terrorist attacks and
illegal immigration combined."]
```

ii Search by ClaimReview author

Another frequent use case is to search for documents which include claim reviews by some author. For example, you may want to only see claim reviews by Snopes:

search?collection=factcheckers&q_schema_org_cr_author_name=Snopes

As with other fields, this type of queries is facilitated by using faceting, e.g.

search?collection=factcheckers&facet.field=schema_org_cr_author_name

iii Search by ClaimReview result (alternative name)

If you want to search for the claim review results, you can use the schema_org_cr_rating_altName field. For example, to get only documents which have been explicitly marked as being “false”, you can use query:

search?collection=factcheckers&q_schema_org_cr_rating_altName=false

Similarly, to search for ratings described as “miscaptioned”:

search?collection=factcheckers&q_schema_org_cr_rating_altName=miscaptioned

Because different fact-checkers use different altNames, you may want to combine searches.

iv Populating a triplestore of ClaimReviews

Although the Content Collector does not provide a triple store, the structured data we crawl can be used to populate such a triple store if required. All the crawled documents which have one or more claim reviews also contain a field schema_org_cr_n3, which contains the raw triples extracted by our custom Nutch plugin. The values in this field adhere to the N3 (Notation3) RDF syntax[15], hence it should be straightforward to iterate through all the results and load a triple store.

An example query to get all the documents with a value for this field is:

search?collection=factcheckers&q_schema_org_cr_n3=*

[15] https://www.w3.org/TeamSubmission/n3/

5. Conclusion and Future Work

In this report we have presented the first version of the Co-Inform Content Collector. This component allows us to:

- Define various types of web sources where disinformation and fact-checking may occur; in particular sources which may be relevant to the Co-Inform pilots.
- Crawl the sources to collect raw text, metadata and structured content.
- Process and enrich raw text in multiple languages, to facilitate automatic understanding of the content in the collected documents.
- Programmatically search the collected and enriched content in a variety of ways that will be useful for inspecting and analyzing whether the content may contain disinformation, dubious claims and/or claim reviews.

The described Content Collector has been built on top of scalable and enterprise-ready components. Initial tests show good performance and high throughput for volumes that can be expected within the project. Should performance issues arise as the project grows, the architecture of this component allows for scaling the solution by adding more machines to the current deployment.

Although we have taken great care in designing the document schema and search API, as the Content Collector is used within Co-Inform, users may request new search functionalities. These will be evaluated on a case-by-case basis and we are open to adding new functionalities when necessary. As new versions of the search API are released, the documentation on our internal wiki will be updated as well as the Swagger specification of the API.

Some improvements we are already considering include:

- Supporting json-ld. The current version of the API only supports plain JSON. However, some of the other components in Co-Inform may benefit from more structured data in the form of json-ld[16] in particular for alignment with a Co-Inform ontology. At the time of writing the Co-Inform ontology is still being finalized, therefore we have not included this as part of the version described in this document.
- Improving claim search. Due to limitations in the way our semantic engine works, the current API only allows users to search for extracted claims by categories in the various taxonomies or by entities. However searching claims by keyword is not currently supported. Thus, while it is currently possible to find claims with category "Social Issues" (in the intelligence taxonomy) and that mention "Donald Trump", it

[16] https://json-ld.org/

is not possible to find claims that match keywords “immigration” and “refugee”. Perhaps more importantly, what is needed is to be able to find similar claims: i.e. to provide a claim and find extracted claims which are similar. We are already working on such functionality, but this is still being researched and will hopefully be added in a second version of the search API. Note that we can already perform keyword search for the schema.org ClaimReviews, this only applies to the extracted claims found during semantic enrichment.

- User configurable sources. The described API only provides a read-only overview of the existing collections, but users cannot add new sources. In the current version, any required sources need to be added manually by the administrators of the Content Collector. The main reason for this is that it is currently unclear whether end users may need this functionality and if so, what will be the volume of such sources. At least for the first phase of Co-Inform, manually adding new sources will allow us to control the quality of the sources as well as the volume of content that needs to be semantically enriched and indexed. If the need arises, we can add a new endpoint to specify new sources semi- of fully automatically.