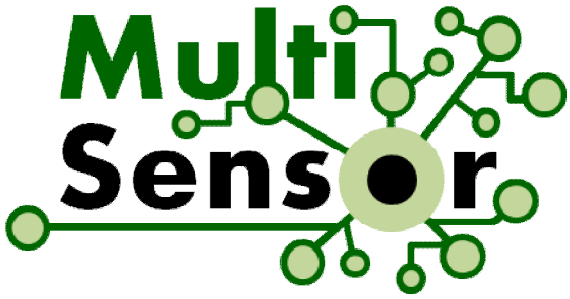

MULTISENSOR

Mining and Understanding of multilinguaL contenT for Intelligent Sentiment Enriched coNtext and Social Oriented inteRpretation

FP7-610411

D7.2

Technical requirements and architecture design

Dissemination level:	Public
Contractual date of delivery:	Month 10, August 31, 2014
Actual date of delivery:	Month 10, August 29, 2014
Workpackage:	WP7 System Development and integration
Task:	T7.1 MULTISENSOR architecture
Type:	Report
Approval Status:	Final Draft
Version:	1.1
Number of pages:	54
Filename:	D7.2_RequirementsArchitecture_2014-08-29_v1.1.pdf

Abstract

This document provides the blueprint for the implementation of the MULTISENSOR platform architecture. It discusses the technical requirements for the platform and its translation into a multi-tier, service-oriented architecture.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	20/07/2014	Document structure	A. Callabed (EVERIS)
0.2	27/07/2014	Initial Comments	S. Vrochidis (CERTH)
0.2	20/08/2014	Draft with integrated inputs	A. Callabed, A. Mas, E. Staromiejski (EVERIS), All partners
0.3	23/08/2014	Comments and inputs	A. Moumtzidou, S. Vrochidis (CERTH)
0.4	27/08/2014	Review ready version	A. Callabed (EVERIS)
0.5	28/08/2014	Review	A. Moumtzidou, S. Vrochidis (CERTH)
1.0	29/08/2014	Final version	A. Callabed (EVERIS)
1.1	29/08/2014	Final updated document	A. Callabed (EVERIS)

Author list

Organization	Name	Contact Information
EVERIS	Axel Callabed	acallabe@everis.com
EVERIS	Alan Mas	alan.mas.soro@everis.com
EVERIS	Enric Staromiejski	enric.staromiejski.torregrosa@everis.com
CERTH	Anastasia Moumtzidou	moumtzid@iti.gr
BM-Y!	Ioannis Arapakis	arapakis@yahoo-inc.com
LINGUATEC	Vera Aleksić	v.aleksic@linguatec.de
UPF	Gerard Casamayor	gerard.casamayour@upf.edu
Ontotext	Kiril Simov	kiril.simov@ontotext.com
pressrelations	Mirja Eckhoff	mirja.eckhoff@pressrelations.de

Executive Summary

This document provides the blueprint for the implementation of the MULTISENSOR platform architecture. It discusses the mapping of user requirements to technical requirements for the platform and its translation into a multi-tier, service-oriented architecture.

A high level view of the logical architecture is presented, together with a breakdown of the different technical modules that make up the platform. The technical specifications for the Use Case applications in terms of structure, processes and workflows are then discussed. Both the shared service API and UC-specific services and subsystems are described.

This specification will be used as the implementation guide for the operational prototype of the platform, the 1st and the 2nd prototype and eventually, the final system. A general discussion on technical conventions, standards and formats is provided, to assist in coordination of efforts between the partners.

Finally, the technical infrastructure layout for deployment of MULTISENSOR is presented. The server layout, configuration and horizontal and vertical scaling strategy for the platform are discussed.

Abbreviations and Acronyms

CMR	Central Multimedia Repository
CNR	Central News Repository
DB	DataBase
DBMS	DataBase Management System
FTP	File Transfer Protocol
FTS	Full-Text Search
HTTP	HyperText Transfer Protocol
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MAF	Multimedia Application Format
MPEG	Moving Picture Experts Group
NER	Named Entity Recognition
NFR	Non-functional Requirement
OPS	OPerationS repository
OWL	Ontology Web Language
RDBMS	Relational DataBase Management System
RDF	Resource Definition Framework
REST	Representational State Transfer
SIMMO	Socially Interconnected and MultiMedia-enriched Object
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol And RDF Query Language
UC	Use Case
UCS	Universal Character Set
UI	User Interface
UTF	UCS Transformation Format
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

Table of Contents

1 INTRODUCTION 8

2 DOMAIN CONCEPTS 9

3 SYSTEM REQUIREMENTS 10

3.1 Methodology 10

3.2 Technical requirements 11
3.2.1 Data collection and storage 11
3.2.2 Data analysis and distillation 13
3.2.3 Search 17
3.2.4 Contributor analysis 19
3.2.5 Users 20
3.2.6 User Interface 21

3.3 Non-functional requirements 23

4 PLATFORM ARCHITECTURE 25

4.1 Logical architecture 25
4.1.1 Harvesting layer 25
4.1.2 Distillation layer 25
4.1.3 Delivery layer 26
4.1.4 Supervisor 26

4.2 Technical architecture 26
4.2.1 High-level view 26
4.2.2 Offline modality 28
4.2.2.1 Supervisor 28
4.2.2.2 Crawler 29
4.2.2.3 Content Extraction pipeline (CEP) 32
4.2.2.4 Social Media Analysis pipeline (SMAP) 38
4.2.2.5 Content alignment pipeline (CAP) 38
4.2.3 Online modality 40
4.2.3.1 Shared business services 40
4.2.3.2 UC1 – Journalism use case 42
4.2.3.3 UC2 – Media Monitoring use case 44
4.2.3.4 UC3 – SME internationalisation use case 46
4.2.4 Data stores 48
4.2.4.1 Central News Repository (CNR) 48
4.2.4.2 Central Media Repository (CMR) 49
4.2.4.3 Semantic Repository (RDF) 49
4.2.4.4 Operations Repository (OPS) 49

4.3 Standards and conventions 49
4.3.1 Exchange formats 49
4.3.2 Encodings 50
4.3.3 Compression 50

4.3.4 Namespaces.....50

5 INFRASTRUCTURE 51

6 CONCLUSION.....53

7 REFERENCES 54

1 INTRODUCTION

This document provides the blueprint for the implementation of the MULTISENSOR platform architecture. It describes the design principles and goals, functional and technical concepts that will be used to guide the iterative development of the system.

MULTISENSOR is envisioned as a platform capable of assisting users in navigating and making sense of large amounts of data broadcast across the entire media spectrum. This broad objective entails several technical challenges that must be considered in the platform architecture design:

- Scalable and timely retrieval and storage of large amounts of data from multiple disparate sources;
- Deep, multi-language semantic processing of multi-modal data for extraction and representation of actionable data;
- Efficient multi-dimensional indexing and retrieval of content for servicing the user-facing applications.
- Agile data visualisation and exploratory interfaces to assist in understanding of information and support decision making processes.

These challenges, together with other, practical realities of the project will drive the design and implementation of the architecture of MULTISENSOR.

Section 2 contains a brief description of lexicon terms used throughout the document.

Section 3 presents technical requirements of the system. It builds on the user requirements and pilot use cases defined as part of Deliverable D8.2 put together by the consortium commercial partners: Deutsche Welle, pressrelations and PIMEC. D8.2 will be cited frequently as the reference source for requirements and descriptions of the usage scenarios.

Section 4 describes the conceptual architecture of the different processes that make up the platform. It also describes the technical architecture of the platform, built on the basis of the requirements D8.2 and the technical roadmap D7.1, put together by the business and technical partners. Deliverable D7.1 will be referenced often as the source for descriptions of the research objectives and the high-level description of technical components. Section 4 also contains a description of the structure, technical implementation and process overview of the Use Case portals that will be built to illustrate the capabilities of the platform.

Then, section 5 describes the plans for the technical infrastructure to host the platform and the strategy for scaling and monitoring as the project evolves.

Finally, section 6 concludes the deliverable.

2 DOMAIN CONCEPTS

Some concepts in the MULTISENSOR lexicon that will be used often in this document are presented below.

SIMMO: Socially Interconnected and MultiMedia-enriched Object. Represents a single, self-contained piece of information captured and aggregated into the MULTISENSOR repositories. A SIMMO contains the main body of text, and associated metadata (such as source, publication date, etc) and all the information generated by the analytical pipelines. A SIMMO maps to an article in the press, or a discrete content unit from social media (e.g. a Tweet, or a Facebook wall post), or a multimedia item such as a video or audio recording.

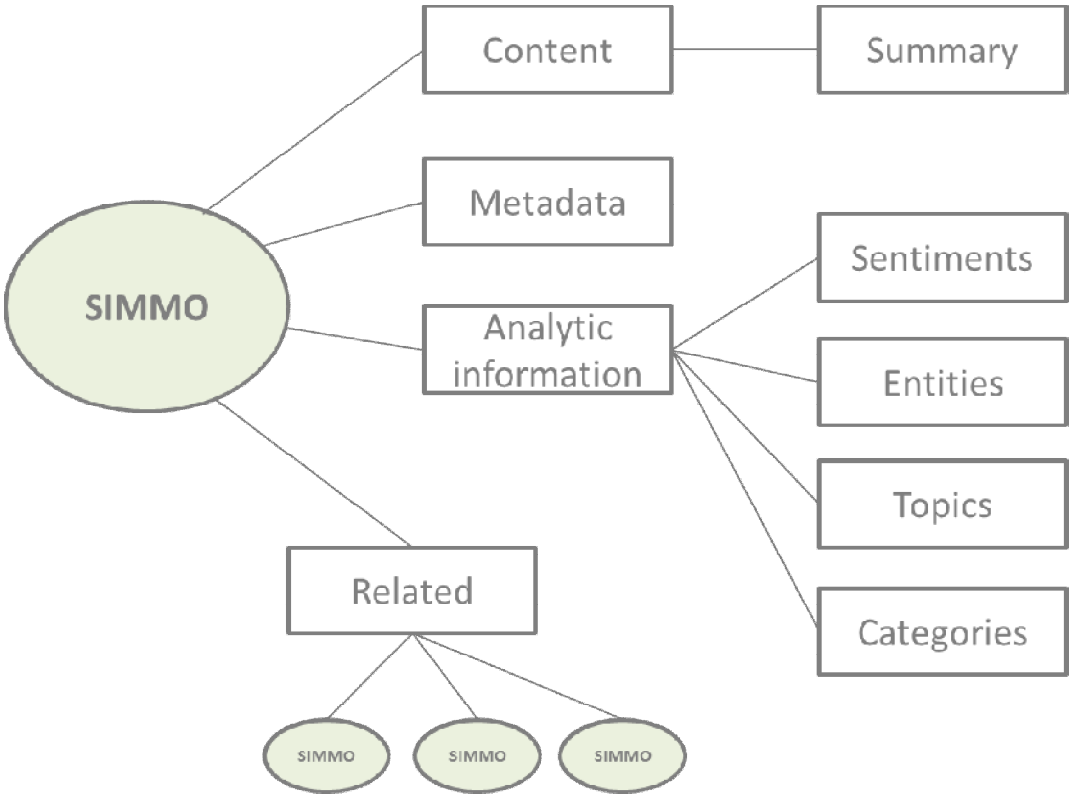

Figure 1: SIMMO graph

SIMMOs are the atomic unit of work in the platform and they are processed individually by the analytic pipelines. Other processes (e.g. social media analysis) process them in bulk for aggregate results and graphs.

INTEL: refers to the set of analytic information generated from a single SIMMO, i.e. the knowledge generated from it. This includes the translations, recognised entities, syntactic analysis, facts, sentiment analysis, inferred information, and so on.

3 SYSTEM REQUIREMENTS

This section presents a discussion of the requirements that drive the design and implementation of the MULTISENSOR platform.

3.1 Methodology

Figure 2 shows the methodology used for requirements management and design of the architecture.

Deliverable D8.2 of the MULTISENSOR project specifies the core User Requirements and a high-level functional description of the Use Cases that will be built to demonstrate the capabilities of the platform and illustrate potential real-world business exploitation scenarios for the technology. The core user requirements are derived from the scientific objectives of the project and the business objectives exposed in the DoW.

Technical requirements ("what the system must do") are synthesised from the user requirements, along with non-functional requirements ("what qualities the system must have"). The logical architecture is then designed to fit all the requirements together, and finally the technical architecture specifies the design for the specific implementation of the logical architecture in terms of technical components.

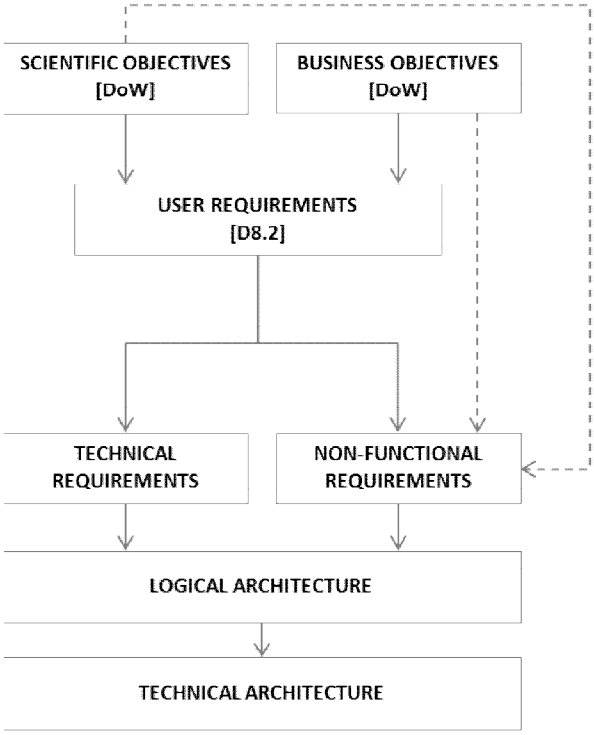

Figure 2: Requirements process

3.2 Technical requirements

The **Technical requirements**[1] describe the specific technical capabilities and features necessary for successful implementation of the user requirements.

The technical requirements are derived from user requirements and translate high-level capabilities or behaviours into actionable technical concepts that drive the design of the architecture and modules of the target system (see Figure 2).

For traceability, in every technical requirement a reference to the source user requirements is presented in the "Related user requirements" section[2]; Forward references to the relevant associated technical components are provided in the "Related technical components / modules" section.

For easier reference, the user requirements of D8.2 can be located at:

- **UC1**: D8.2, pp. 20-22.
- **UC2**: D8.2, pp. 33-35.
- **UC3**: D8.2, pp. 43-45.

Based on the initial user requirements, the final mapping to technical requirements is presented in the tables below.

3.2.1 Data collection and storage

R-DATA-1	**Automated collection of news and multimedia content from online and print media**
Description	The platform must aggregate content from multiple news sources and consolidate it into a central repository for analysis. This process will be continuous, regularly polling all content sources for new content to be collected. Structured content sources (APIs) providing access to news will be used.
Related User Requirements	UC1: User defined search UC2: User defined search UC3: User defined search
Related technical components/modules	Supervisor, Crawler, Media collector (PR)

R-DATA-2	**Automated collection of news and multimedia content**

[1] There are multiple conflicting nomenclatures and methodologies in requirements management. In some cases, "technical requirements" is synonymous to non-functional requirements; in other cases, "system requirements" or "functional requirements" are used to describe this concept. In the context of MULTISENSOR, for clarity and consistency with the DoW, *technical requirements* are to be understood literally as technical requirements.

[2] Technical requirements implicitly derived from other technical requirements are not mapped to user requirements but included for completeness.

	from targeted sites
Description	The platform must aggregate content from designated online sources and consolidate it into a central repository for analysis. This process will be continuous, regularly polling all content sources for new content to be collected. A mix of pre-built and ad hoc crawlers and APIs will be used to scrape the content.
Related User Requirements	UC1: User defined search UC2: User defined search UC3: User defined search
Related technical components/modules	Supervisor, Crawler, Site collector (Y)

R-DATA-3	**Automated collection of news and multimedia content from social media**
Description	The platform must aggregate content from social media (Facebook, Twitter, Google+) and consolidate it into a central repository for analysis. This process will be continuous, regularly polling all content sources for new content to be collected. Public APIs provided by the social networks will be used.
Related User Requirements	UC1: User defined search UC2: User defined search UC3: User defined search
Related technical components/modules	Supervisor, Crawler, Media collector (PR)

R-DATA-4	**Scalable storage of source news and metadata**
Description	The system must be able to store potentially large amounts of source news material in plaintext, and be able to grow in scope and magnitude as new sources are added. Metadata for news must be stored and searchable both via FTS and structured queries.
Related User Requirements	UC1: User defined search UC2: User defined search UC3: User defined search
Related technical components/modules	Central News Repository (CNR)

R-DATA-5	Scalable storage of source multimedia content and metadata
Description	The system must be able to store potentially large amounts of source multimedia material, and be able to grow in scope and magnitude as new sources are added. Metadata for multimedia files must be stored and searchable.
Related technical components/modules	Central Media Repository (CMR)

R-DATA-6	Scalable storage of source multimedia content and metadata
Description	The system must be able to store and index potentially large amounts of data in RDF format and support integration of multiple supporting ontologies and datasets. The RDF repository must be live writable and must support reasoning. RDF data must be searchable using SPARQL or other means.
Related technical components/modules	Central News Repository (CNR)

R-DATA-7	Scalable storage of structured and semi-structured data
Description	The system must be able to store potentially large amounts of structured and semi-structured data. Due to the largely unstructured nature and loose relationships of the source material in MULTISENSOR and expected volume of data, a NoSQL solution is preferred over a traditional RDBMS.
Related technical components/modules	Operations Repository (OPS)

3.2.2 Data analysis and distillation

R-DIST-1	Automatic detection of language from arbitrary texts
Description	The system must be able to automatically determine the source language of an arbitrary piece of text. Supported languages will be German, French, English, Spanish and Bulgarian.
Related User Requirements	UC1: Automatic language detection and translation UC2: Automatic language detection and translation UC3: Automatic language detection and translation

Related technical components/modules	Language detection Service [WP2]

R-DIST-2	**Machine translation of arbitrary texts**
Description	The system must be able to automatically translate an arbitrary text to a specified language. Supported languages will be German, French, English, Spanish and Bulgarian: any of the possible combinations.
Related User Requirements	UC1: Automatic language detection and translation UC2: Automatic language detection and translation UC3: Automatic language detection and translation
Related technical components/modules	Translation Service [WP2]

R-DIST-3	**Speech-to-text analysis of audio files**
Description	The system must be able to perform textual transcriptions of arbitrary audio fragments containing speech. Supported source languages will be German, French, English, Spanish and Bulgarian.
Related User Requirements	UC1: Automatic language detection and translation UC2: Automatic language detection and translation UC3: Automatic language detection and translation
Related technical components/modules	Speech recognition (ASR) Service [WP2]

R-DIST-4	**Extraction of entities from arbitrary texts**
Description	The system must be able to recognise and extract named entities from arbitrary texts (source news, social media posts and threads, speech-to-text transcripts) and store them. Offsets of occurrences in the text must be kept for future reference.
Related User Requirements	UC1: Named Entities Extraction UC2: Named Entities Extraction
Related technical components/modules	NLP Preprocessing Service [WP2], NER Service [WP2], Dependency parsing Service [WP2], Content extraction Service [WP6]

R-DIST-5	Sentiment analysis of arbitrary texts
Description	The system must be able to determine the sentiments regarding entities in a previously parsed and annotated text (source news, social media posts and threads, speech-to-text transcripts). Sentiment information must be stored and searchable, and linked to the source material for future reference.
Related User Requirements	UC1: Sentiment analysis UC2: Sentiment analysis UC3: Sentiment analysis
Related technical components/modules	Sentiment analysis Service [WP3]

R-DIST-6	Extractive summarisation of arbitrary texts
Description	The system must be able to generate short plaintext summaries from arbitrary texts. Summaries will be assembled by selecting relevant fragments from the original text, with no additional input. Generated summaries must be stored and searchable.
Related User Requirements	UC1: Automatic summarisation UC2: Automatic summarisation UC3: Summarisation
Related technical components/modules	Extractive summary Service [WP6], Summarisation service

R-DIST-7	Abstractive summarisation of arbitrary texts
Description	The system must be able to generate plaintext summaries from arbitrary texts. Summaries will be assembled from semantic annotations and linguistic information extracted from the original text. Generated summaries must be stored and searchable.
Related User Requirements	UC1: Automatic summarisation UC2: Automatic summarisation UC3: Summarisation
Related technical components/modules	Extractive summary Service [WP6], Summarisation service

R-DIST-8	Visual concept extraction from video

Description	The system must be able to analyse a raw video stream and extract relevant concepts and events according to specific taxonomies and categories. Extracted concepts and events must be stored and searchable, and linked to the original video.
Related User Requirements	UC1: User defined search UC1: Enrichment UC3: Image-based understanding UC3: Correlation
Related technical components/modules	Concept and event detection Service [WP4]

R-DIST-9	**Automatic content classification**
Description	The system must be able to automatically classify content and related analytic information according to multiple pre-defined categories (topics), based on the linguistic and visual analysis of the source material.
Related User Requirements	UC2: Topic detection and clustering
Related technical components/modules	Classification Service [WP4], Context extraction Service [WP3], Indexing Service [WP4]

R-DIST-10	**Multi-modal content indexing**
Description	The system must be able to efficiently index and retrieve content (text and multimedia) and related analytic information according to multiple dimensions for rich search and mining capabilities. The index must be able to scale to a potentially very large dataset (hundreds of millions of items) with reasonable performance.
Related User Requirements	UC1: User defined search UC1: Context extraction UC1: Enrichment UC1: Visualisations UC2: Semantic search UC2: Visualisations UC3: Semantic search UC3: Visualisations
Related technical	Indexing Service [WP4], Indexing service

components/modules	

R-DIST-11	Content alignment and consistency checking
Description	The system must periodically process the harvested facts and data and detect contradictory statements, inconsistencies and corrupt data to ensure integrity of the repository.
Related User Requirements	N/A
Related technical components/modules	Content alignment pipeline (CAP)

3.2.3 Search

R-SCH-1	Semantic search
Description	The system must be able to search for keywords or phrases across the entire dataset with reasonable performance. Apart from traditional FTS features (stemming, noise word removal, tokenisation, etc.), the system must provide semantic search capabilities such as support for synonyms and hyponyms, fuzzy matching and user-based relevance filtering. The system must also support search disambiguation of terms by the user.
Related User Requirements	UC1: User defined search UC2: Semantic search UC3: Semantic search
Related technical components/modules	Indexing service, Semantic search Service

R-SCH-2	Flexible search filtering
Description	The system must support filtering of search results by any property.
Related User Requirements	UC1: User defined search UC2: Semantic search UC3: Semantic search
Related technical components/modules	Indexing service, Semantic search Service

R-SCH-3	Search facet support

Description	The system must support automatic aggregation of search results into facets for flexible search filtering and clustering. Facets must be constructed from the intrinsic properties of the content (e.g. “source”, “country”, etc.), the analytic information results (e.g. entities) and the automatic classification categories.
Related User Requirements	UC1: User defined search UC2: Semantic search UC3: Semantic search
Related technical components/modules	Indexing service, Semantic search Service

R-SCH-4	**Related content search**
Description	The system must support searching for content related to a given item (e.g. other items on the same topic, other items by same author, more news in same location…).
Related User Requirements	UC1: Enrichment UC3: Image-based understanding
Related technical components/modules	Similarity search Service

R-SCH-5	**Reference data search**
Description	The system must support querying for reference information about EU countries such as demographics, macro-economic and business sector data. Reference data will be collected from public sources and statically incorporated into the data repositories.
Related User Requirements	UC3: Correlation UC3: Comparison and decision support
Related technical components/modules	UC3 Reference data service

R-SCH-6	**Risk assessment**
Description	The system must support risk assessment, based on indicators calculated from reference data. Given two entities with comparable indicators, the system must identify gaps and opportunities to aid in decision making.
Related User Requirements	UC3: Correlation

	UC3: Comparison and decision support
Related technical components/modules	UC3 Decision support Service

R-SCH-7	**Query-by-image search**
Description	The system must support searching for images similar to a given image.
Related User Requirements	UC1: Enrichment UC3: Image-based understanding
Related technical components/modules	Similarity search Service

R-SCH-8	**Saved searches**
Description	The system must support storing a specific search phrase plus any associated filters for future re-execution. Saved searches are labelled and stored for a specific user profile.
Related User Requirements	UC1: Configurability of system results UC3: Automatic search
Related technical components/modules	UC1 Profile service, UC2 Profile service, UC3 Profile service, Semantic search Service

R-SCH-9	**User-based relevance filtering**
Description	The system must support flagging of results as irrelevant for a particular saved search. Once an item has been marked as irrelevant, it is blacklisted and will be filtered out of future runs of the specific saved search.
Related User Requirements	UC1: Configurability of system results UC2: User-based validation of relevance
Related technical components/modules	UC1 Profile service, UC2 Profile service, UC3 Profile service, Semantic search Service

3.2.4 Contributor analysis

R-CONT-1	**Contributor tracking**
Description	The system must store information about authors of content incorporated in the platform, when available (Twitter users, journalists, etc.). This information must be searchable.

Related User Requirements	UC1: Contributor analysis UC1: Visualisations UC2: Contributor analysis UC2: Visualisations
Related technical components/modules	Social Media Analysis pipeline (SMAP), Social graph Service

R-CONT-2	**Contributor network detection**
Description	The system must track Twitter activity for contributors and calculate the degree of influence in their network.
Related technical components/modules	Social Media Analysis pipeline (SMAP), Social graph Service

3.2.5 **Users**

R-USR-1	**User profiles**
Description	The system must support discrete user accounts to store preferences, saved searches and any other per-user specific information.
Related User Requirements	UC1: GUI development UC2: GUI development UC3: GUI development
Related technical components/modules	UC1 Profile service, UC2 Profile service, UC3 Profile service

R-USR-2	**User profile authentication**
Description	The system must support authentication for accessing user profiles and accessing the applications. Authentication using a Google account must be supported.
Related User Requirements	N/A
Related technical components/modules	UC1 Profile service, UC2 Profile service, UC3 Profile service

R-USR-3	**User favourites/bookmarks**
Description	The system must support users flagging items as “favourites” for future reference.
Related User Requirements	UC1: Configurability of system results

	UC1: GUI development UC2: GUI development UC3: GUI development
Related technical components/modules	UC1 Profile service, UC2 Profile service, UC3 Profile service

3.2.6 User Interface

The user requirements in D8.2 contain mock-ups of tentative user interfaces for the Use Case applications. Here, the synthesised technical requirements from those mock-ups are presented.

R-UI-1	**Journalism application**
Description	The system must implement a web application following the tentative behaviour and mock-ups described in D8.2, section 5.1.
Related User Requirements	UC1: GUI development
Related technical components/modules	UC1 – Journalism use case

R-UI-2	**Media monitoring application**
Description	The system must implement a web application following the tentative behaviour and mock-ups described in D8.2, section 5.2.
Related User Requirements	UC2: GUI development
Related technical components/modules	UC2 – Media Monitoring use case

R-UI-3	**SME internationalisation support application**
Description	The system must implement a web application following the tentative behaviour and mock-ups described in D8.2, section 5.3.
Related User Requirements	UC3: GUI development
Related technical components/modules	UC3 – SME internationalisation use case

The following are significant UI components required for implementation of the three UC applications:

R-UI-4	**Multi-language**

Description	The UI of the UC applications must be multi-language and hot-switchable by the user. Control literals, labels and navigation elements must support multiple languages (list of languages TBD).
Related User Requirements	D8.2, section 5.1
Related technical components/modules	UC1 – Journalism use case, UC2 – Media Monitoring use case, UC3 – SME internationalisation use case

R-UI-5	**Dynamic sortable lists**
Description	The system must support display of collections of items in a traditional list format with columns, and allow for sorting on column values. Lists must support paging for good performance with large result sets.
Related User Requirements	UC1: Visualisations UC2: Visualisations UC3: Visualisations
Related technical components/modules	UC1 – Journalism use case, UC2 – Media Monitoring use case, UC3 – SME internationalisation use case

R-UI-6	**Dynamic hierarchical lists**
Description	The system must support display of hierarchical datasets in a tree-like format (e.g. news items, grouped by category, with nested category concepts).
Related User Requirements	UC2: Visualisations
Related technical components/modules	UC1 – Journalism use case, UC2 – Media Monitoring use case, UC3 – SME internationalisation use case

R-UI-7	**Tag clouds**
Description	The system must support display of datasets in tag cloud format, with size used as an indicator for frequency/relevance/importance, depending on context.
Related User Requirements	UC1: Visualisations UC2: Visualisations UC3: Visualisations
Related technical components/modules	UC1 – Journalism use case, UC2 – Media Monitoring use case, UC3 – SME internationalisation use case

R-UI-8	Charts
Description	System must support rich visualisations of data (pie charts, bar charts, line charts) to provide simple analytics capabilities. Specific requirements for data to be rendered are not yet clear: however, but influence network graphs, frequency analysis and distributions are foreseen.
Related User Requirements	UC1: Visualisations UC2: Visualisations UC3: Visualisations
Related technical components/modules	UC1 – Journalism use case, UC2 – Media Monitoring use case, UC3 – SME internationalisation use case

3.3 Non-functional requirements

While functional requirements capture the *behaviour* of a system, **non-functional requirements** (NFR) are a set of *qualities* that are valuable to stakeholders and *constraints* that affect planning, design and architecture decisions in different ways.

The following NFR are in effect for the MULTISENSOR platform:

Service oriented: the platform will be built as an orchestration of discrete pieces of functionality, exposed as services to the rest of the system. Services will communicate with each other via the HTTP protocol, using the REST architectural design pattern[3].

Open standards: data will be serialised and exchanged using open, well-known formats and standards, such as XML, JSON, RDF and their associated dialects. Open standards are easy to parse, easy to debug, are well understood and have lots of tools that support them.

Interoperable: functionality will be exposed as an API of services accessed by HTTP; the underlying technology of each service implementation will be hidden in such a way that it could be changed in the future without any impact on the rest of the system. Services will not expose implementation details and operate on simple, platform-independent data transport objects.

Scalable: the platform needs to be designed in such a way that it can be scaled to deal with increasing amounts of data and users. No single element can compromise the ability of the platform as a whole to grow.

Reusable: where possible, discrete pieces of functionality of the platform should be reusable, i.e., they should have the least possible amount of dependencies on other elements, and be able to operate with little or no shared context. Services will be designed to be self-descriptive and self-contained, where possible.

Location-independent: the platform will run on a logically and geographically distributed infrastructure; there will be a mix of processes running in partners' premises and processes

[3] See **http://www.w3.org/2005/Talks/1115-hh-k-ecows/#(1)** for a discussion of the REST architectural pattern and comparison with other RPC methods.

running in cloud infrastructure. The platform will allow for transparent orchestration of these resources and relocation of individual elements as required.

Iterative: the architecture will allow for iterative or evolutionary development. Starting from a backbone of placeholder and dummy services which will be put in place early in the project, additional capabilities and increasing functionality will be added without major changes to the overall design of the platform. Having a working end-to-end solution very early helps detect problems early, provides everyone with a complete vision from the beginning, and helps evaluate progress by providing tangible, measurable improvements over time.

Simple: as a general design principle, the platform as a whole should be easy to understand, both in terms of processes and topology. Service and pipeline design will be built on the "principle of least surprise[4]": interactions, dependencies and side effects will be kept obvious.

[4] See **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Principle_of_least_astonishment.html** and **http://www.catb.org/~esr/writings/taoup/html/ch01s06.html#id2878339** for background discussion on this principle.

4 PLATFORM ARCHITECTURE

This section describes the architecture design of the MULTISENSOR platform.

The **logical architecture** (see section 4.1) describes the high-level technical processes and flows, based on the user requirements and global research objectives. It provides an abstract, technology-agnostic view of the platform and the flows of information.

The **technical architecture** (see section 4.2) describes the specific technical implementation of the logical architecture, in terms of components, services and data stores. It provides the reference model for implementation of the platform.

4.1 Logical architecture

The logical architecture of the MULTISENSOR platform (see Figure 3) is comprised of three major subsystems that correspond to stages in the value generation chain: the harvesting layer, the distillation layer and the delivery layer. Additionally, the supervisor component glues the layers together and orchestrates the data flows and execution.

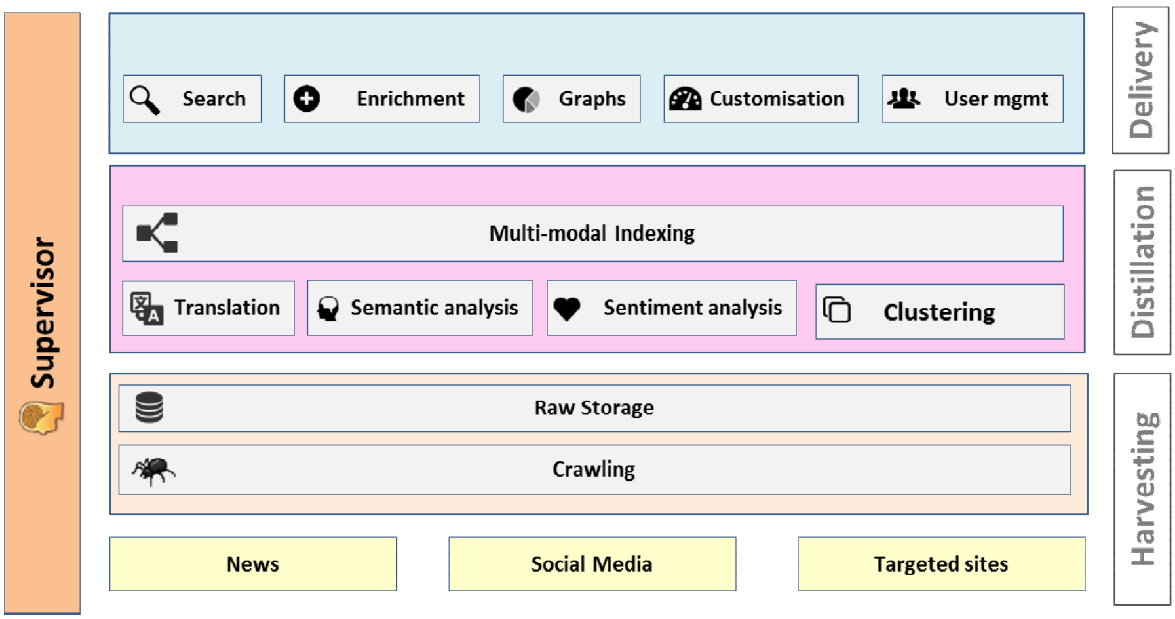

Figure 3: MULTISENSOR logical architecture

4.1.1 Harvesting layer

The Harvesting Layer is responsible for capturing and aggregating multi-language, multi-source, multi-media content from a variety of sources into the platform, for analysis and exploitation. The data collected at this stage is the basis for operation of the platform.

4.1.2 Distillation layer

The Distillation Layer comprises of a set of processes which analyse the harvested information and generate actionable knowledge (INTEL) from it. Analytical processes are the core of the platform and include linguistic analysis, inference, multi-dimensional clustering and integration, social graph detection, summarisation, and more. There are analytic processes operating both on single, discrete pieces of content, and on the entire dataset.

4.1.3 Delivery layer

The Delivery Layer implements interfaces for users to interact with the platform. It is responsible for hosting applications and services that build value from the information in MULTISENSOR to implement the Use Cases.

4.1.4 Supervisor

The Supervisor system monitors and orchestrates execution of the different subsystems of the platform. It is responsible for scheduling background and offline processes, monitoring health of the system and providing other auxiliary services.

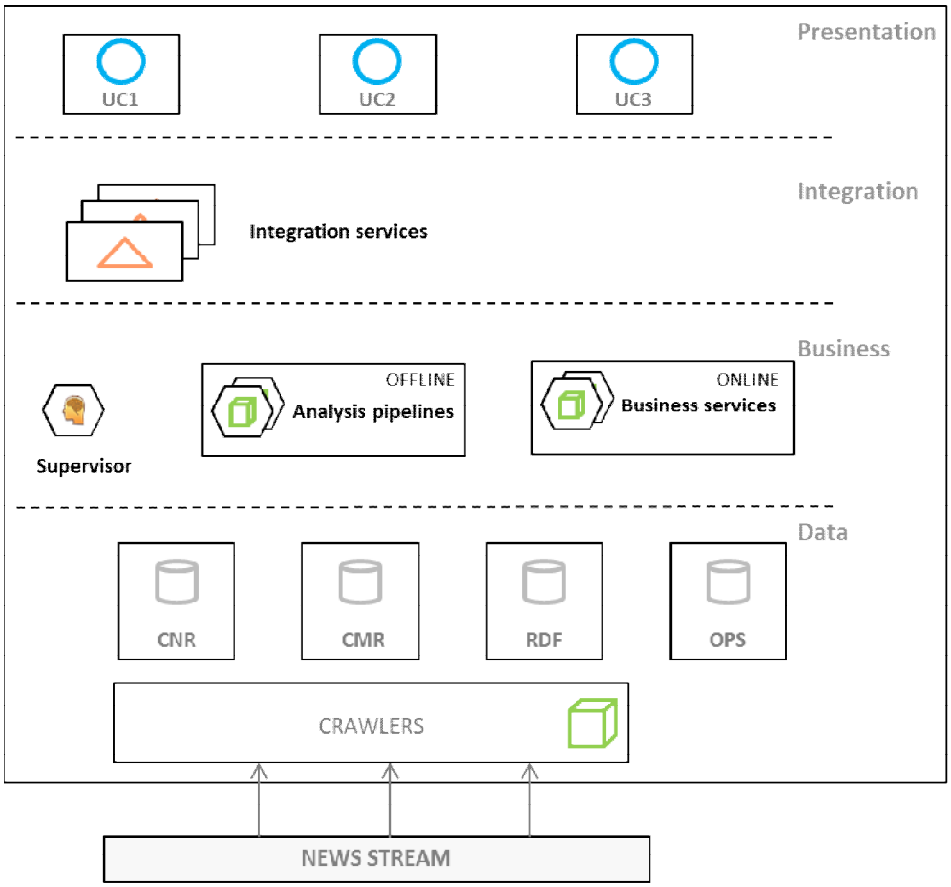

Figure 4: MULTISENSOR architecture high-level view

4.2 Technical architecture

4.2.1 High-level view

Layered architecture

The technical architecture of the platform (Figure 4) is modelled following the well-known and field tested N-tier architecture pattern[5]. This model proposes logical and physical

[5] For general discussions on this pattern, see for example **http://en.wikipedia.org/wiki/N-tier_architecture** or **http://msdn.microsoft.com/en-us/library/ee658117.aspx#NTier3TierStyle**

separation of concerns (data access, business logic and presentation) in layers, with an emphasis on maintainability and decoupling of components.

The separation in *layers* (logical separation) and/or *tiers* (physical separation) allows for separate evolution, or even replacement, of the different components of the system, with changes in one layer not compromising other layers. Physical tiers provide the extra benefits of increased security and system resiliency to failure, which are crucial in large scale and mission-critical enterprise systems.

An N-tier architecture fits well for the loosely coupled, distributed, service-heavy development model of MULTISENSOR. Separating presentation from processing logic helps reusability, as new, independent clients can be built for interaction with the core system, or specific logic modules can be refitted into existing systems.

The **presentation layer** is responsible for displaying information to clients. In the case of MULTISENSOR, "clients" correspond to human users. This is typically some form of User Interface (UI) the users can interact with. The UI collects input from the user and presents results of processing that takes places in the lower levels of the architecture. Requests are sent to the business layer through integration services for execution.

In the MULTISENSOR platform, this corresponds to three discrete web applications providing an interface to the Use Cases created for demonstration of the system's capabilities. It maps to the Delivery layer in the logical architecture.

The **integration layer** acts as middleware between the Presentation and Business layers, and performs any auxiliary functions required by the application design, e.g. caching, proxying, format conversions, queuing of requests, etc. They also act as wrappers for some business servers. The Integration layer maps to the Delivery layer in the logical architecture.

The **business layer** is the core of the platform, and is where the actual processing logic takes place. Business logic manipulates data from the data layer and sends it "upwards" for display to clients, or "downwards" for storage of computation results. In the MULTISENSOR platform, the business layer contains all analytic processes, and search functionalities.

The business layer maps to the Distillation layer of the logical architecture.

The **data layer**'s responsibility is to provide access to all data repositories of the platform. In the MULTISENSOR platform, the data layer contains all the databases storing harvested data and the analytic information extracted from it.

Modalities

The MULTISENSOR platform is a dynamic system, designed to process data and generate new intelligence as it is generated in the media and social networks it monitors. As a result of this, the platform hosts 2 interconnected subsystems, referred to as "modalities": the **offline** modality, and the **online** modality.

The offline modality (see section 4.2.2) is asynchronous and unattended, and is responsible for collection and analysis of content as it becomes available; the online modality (see section 4.2.3) is responsible for the live exploitation of the data and the execution of the applications that sit on top of the backend.

4.2.2 Offline modality

The offline modality is responsible for the collection of content for the platform, and asynchronous execution of the heavy duty analysis work that would be unfeasible to do live.

Due to the experimental and resource-intensive nature of the analytic processes involved in processing of the source material, a global design strategy of the platform is to attempt to do as much as possible offline, and then use pre-calculated/pre-digested data for the online modality (i.e. the user-facing web applications).

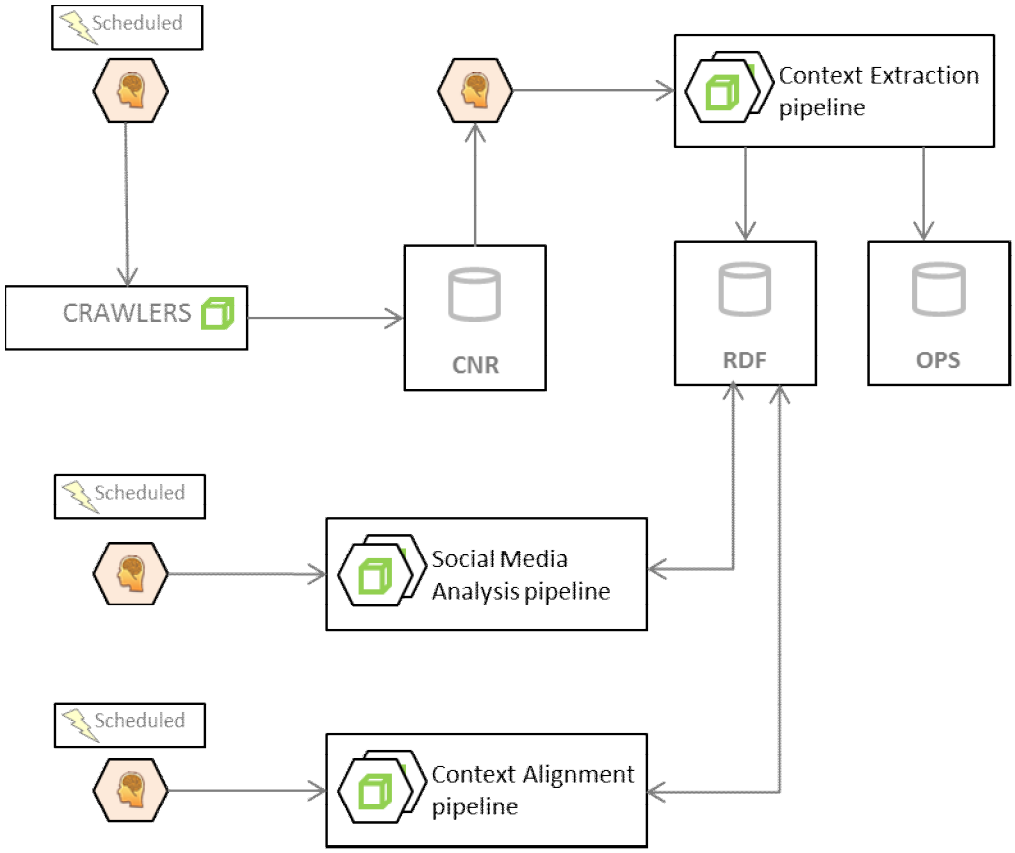

Figure 5: Offline modality overview

4.2.2.1 Supervisor

The supervisor is responsible for scheduling, orchestration, and execution of the data harvesting and analytical processes in the platform.

Scheduler

The scheduler executes jobs at specified times or intervals, and logs the results of the execution. It is analogous to the traditional `cron` tool in UNIX systems[6].

The following scheduled jobs are foreseen:

- Regular execution of crawler jobs (see section 4.2.2.2);
- Execution of CEP pipeline for every incoming news item (see Content Extraction pipeline (CEP));

[6] See **http://en.wikipedia.org/wiki/Cron** for background on cron.

- Periodic execution of the Content Alignment Pipeline (see Content alignment pipeline (CAP));
- Periodic execution of the SMAP pipeline (see Social Media Analysis pipeline (SMAP));

Configuration repository

The supervisor exposes a service to access configuration of the platform as key-value pairs. This allows services to obtain configuration values from a centrally-maintained, updated repository to prevent configuration files in multiple systems going stale or missing essential information.

Pipeline driver

The supervisor is responsible for initiating and orchestrating calls to the pipelines that form the backbone of the Distillation layer. It chains the calls to the services and ensures proper execution of the process. It also collects information about execution time and error ratios, to help identify bottlenecks or problematic areas in the pipelines.

Health monitor

The supervisor monitors the state of the different services of the MULTISENSOR platform to ensure they're operational, by sending PING requests. If a service is found to be offline, alerts are generated.

Implementation: the Supervisor component is built as a Node.js application.

4.2.2.2 Crawler

The Crawler component (Figure 6) is the process responsible of collecting source material for use by the MULTISENSOR platform. It runs regularly (initiated by the Supervisor), going over a set of content sources, and sends the retrieved material through the analysis pipeline for extraction of intelligence.

The Crawler engine is a plugin-based system. It provides a generic interface to crawl a content source, and store the result in the Central News Repository. Plugins (called "collectors") provide the functionality to do the actual crawling of a resource and return the raw data.

Two collector plugins are foreseen for the development of MULTISENSOR: the **Media collector** and the **Site collector**.

Once content is delivered by a collector, every item is run through the Aggregator, which does processing on the item:

- Checks that it's long enough for analysis purposes (e.g. articles containing a single link or a single word are discarded). Items found to be unusable are automatically discarded.
- Verifies that it contains the source URL.
- Assigns a unique ID.
- Does basic integrity checking (validation of dates, etc).
- Converts the item to the common JSON format for SIMMOs.

After being piped through the Aggregator, items are stored in the Central News Repository (for text items) and the Central Multimedia Repository (for multimedia items).

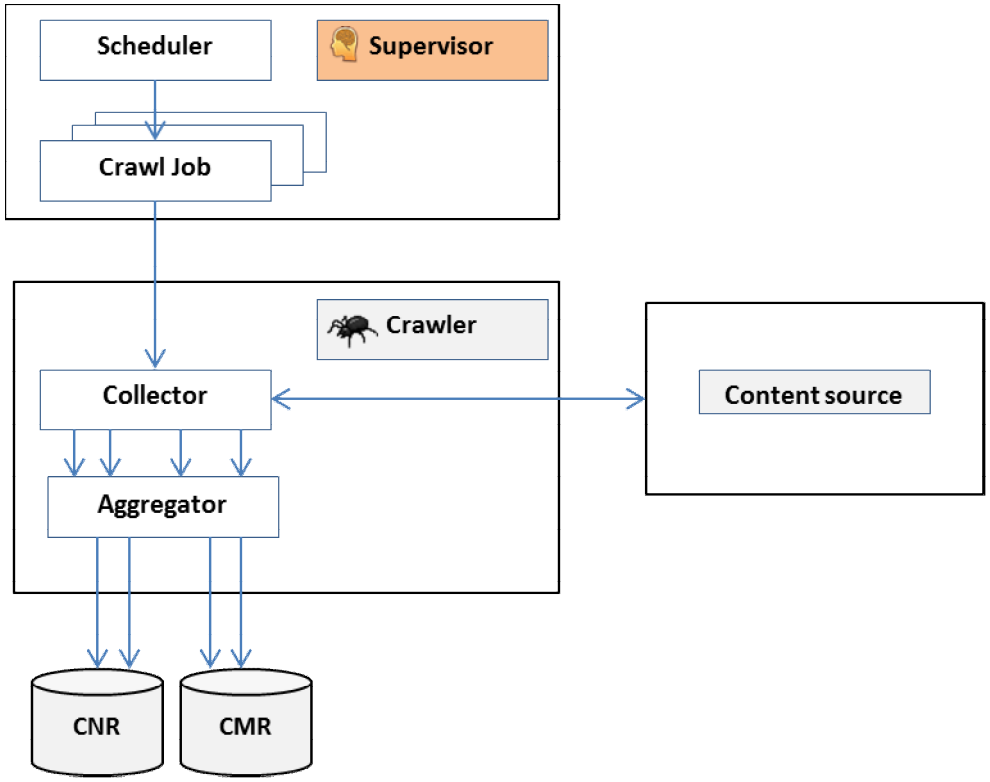

Figure 6: Crawler process

Media collector (PR)

The Media Collector (called the "PR" collector for short) (Figure 7) retrieves content from pressrelations NewsRadar system[7], which aggregates news from media all over the world (print media, online media and social networks) using proprietary technology by PR.

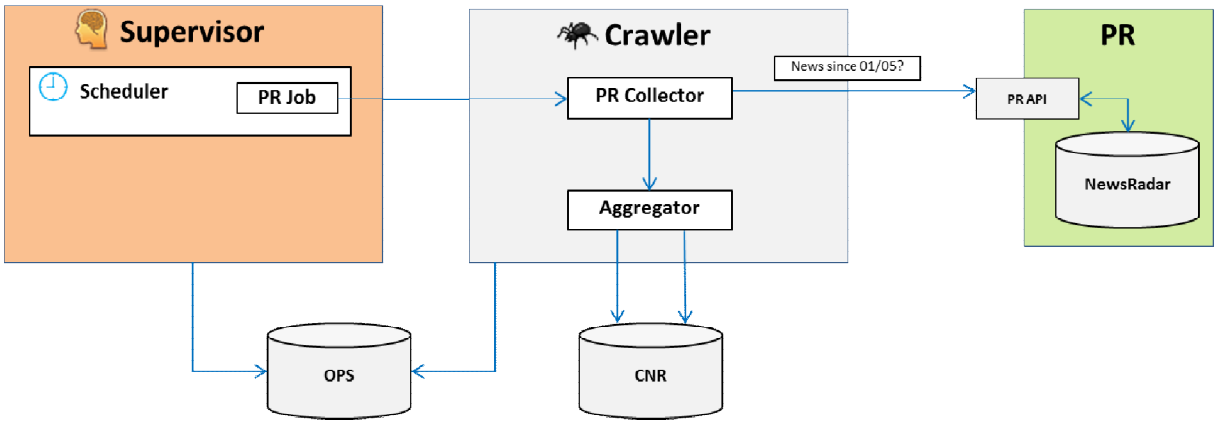

Figure 7: PR collector

NewsRadar is accessed through a JSON-based API that delivers data which has already been preselected as relevant for the topics of interest for MULTISENSOR (i.e. referring to the Use Cases defined in D8.2). Content is retrieved incrementally as new articles are available.

An example of the JSON response for a PR API request is shown in Figure 8.

```
{
```

[7] http://www.pressrelations.com/pressrelations/index.cfm/en/newsradar

```
  "results": [
     {
      "use_case": "UC1 - Household Appliances",
      "language": "en",
      "country": "INT",
      "_id": "",
      "crawled": "2014-06-17",
      "multimediaUrls": [],
      "source": "facebook.com",
      "pr_summary": "",
      "date_timestamp": 1402992610,
      "body": "Lorem ipsum dolor sit amet, consectetur adipiscing elit",
      "feed": "",
      "date": "2014-06-17",
      "article_id": 1372557200,
      "pr_feed": "internet",
      "url": "http://www.example.com?story_fbid=245151655673882&id=129981993857516",
      "title": "Video: Whirlpool India"
    },
        {
      "use_case": "UC1 - Energy Policy",
      "language": "de",
      "country": "DE",
      "_id": "",
      "crawled": "2014-06-16",
      "multimediaUrls": [
        { "multimedia_url": "http://www.example.com/1" },
        { "multimedia_url": " http://www.example.com/1" },
        { "multimedia_url": " http://www.example.com/1" },
      ],
      "source": "Die Zeit",
      "pr_summary": "",
      "date_timestamp": 1402920008,
      "body": "Proin adipiscing sit amet ligula nec dapibus. Morbi non felis vel mi
tempus sagittis. Proin aliquam non purus in eleifend. Fusce feugiat, nisl vitae vulputate
laoreet, augue lorem mattis urna, eget mattis lectus libero eu justo. Donec eget
pellentesque nisl. Proin cursus mauris a suscipit dapibus",
      "feed": "",
      "date": "2014-06-16",
      "article_id": 1371797639,
      "pr_feed": "internet",
      "url": "http://www.zeit.de/wirtschaft/2014-06/ukraine-russland-gasstreit-
lieferstopp/komplettansicht",
      "title": "Ukraine-Krise Russland stoppt Gaslieferungen an Ukraine (11:59, ZEIT
ONLINE)"
    }
  ],
  "errors": [],
```

```
  "nextPageUrl": "/api/news/internet?count=10&maxId=1372650042&customerKey=?????"
}
```


Figure 8: PR API response

Site collector (Y)

The Site Collector (called the "Y Collector" for short) (Figure 9) aggregates content crawled by the targeted crawler.

The targeted crawler process (developed as part of WP7) will crawl content from designated websites to extract information, and store it locally. The targeted crawler will have a generic scraper capable of limited scraping of arbitrary websites, and ad-hoc crawlers for high-precision scraping of designated websites.

The targeted crawler will expose the crawled content via a JSON API the collector will use to fetch data, similarly to the process of the PR collector.

Implementation: the crawler infrastructure and PR collectors are Node.js applications. The targeted crawler is built as a Hadoop/ HBase job, using a combination of custom Java and Python tools and Python such as Nutch[8] and Boilerplate.

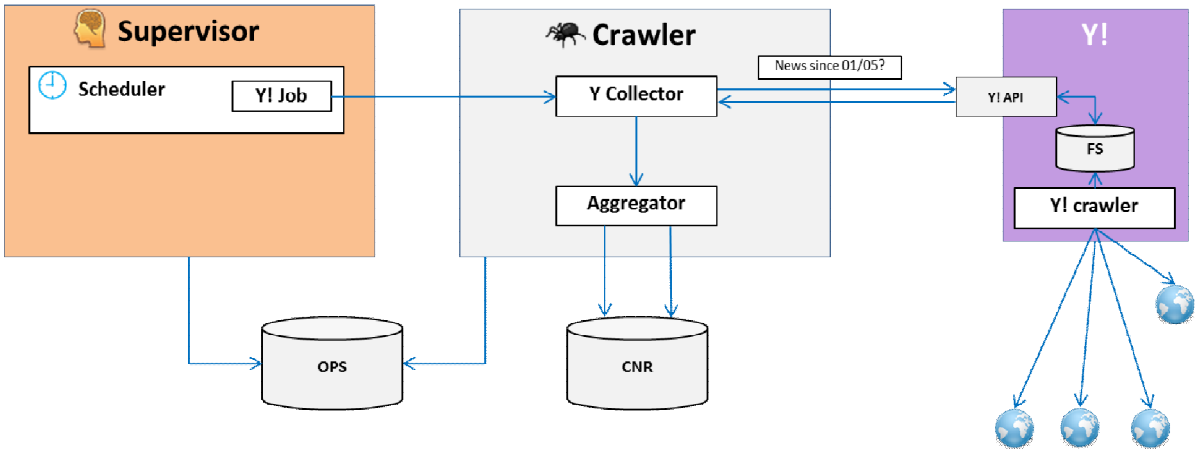

Figure 9: Y Collector

4.2.2.3 Content Extraction pipeline (CEP)

The Content Extraction pipeline (CEP) (Figure 10) is the main analytical process for all SIMMOs harvested by the Platform.

The CEP chains execution of several language analysis services to generate intelligence from the contents of a text article or a video segment, from base syntactical analysis to sentiment analysis to automated clustering and classification of the contents.

The Pipeline Driver in the Supervisor fetches a predefined number of items from the Central News Repository, requesting those that are marked as unprocessed. Each one of them is sequentially sent down the pipeline, as shown in Figure 10.

The CEP process uses a JSON container to incrementally store the results of the different analysis steps: every service receives this JSON container as input, and can inspect data generated by previous services in the chain. It does the processing and appends the output

[8] See **http://nutch.apache.org**

of its work to the JSON container, and returns it. At the end of the pipeline, the JSON container holds the complete results of the analysis.

An example JSON container for a news article is shown in Table 1.

Header Title, body and metadata


```
"_id"      : "bfb7759a67daeb65410490b4d98bb9da7d1ea2ce",
"title"    : "Lorem ipsum",
"source"   : "Random publication",
"url"      : "http://example.com/foo/26726762",
"country"  : "de",
"date"     : "1406622688",
"crawled"  : "1406626217",
"body"     : "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla mauris sem, blandit nec dignissim vitae, tincidunt id enim. Praesent pellentesque, elit ut ultrices fringilla, ipsum magna porttitor nulla, vel pellentesque velit mi et magna. Etiam laoreet, nulla a tristique dignissim, arcu orci pellentesque est, ut blandit magna diam pellentesque felis. Quisque adipiscing mi ac odio consequat lacinia. Morbi sed laoreet diam, id condimentum mauris. Nullam nec neque eleifend, vestibulum erat egestas, commodo eros. In et sagittis tortor, quis sagittis eros. Nullam vel arcu ut neque tempus sollicitudin vitae in libero. Nullam volutpat dui nec blandit eleifend.",
"pr_feed" : "news",
"multimediaUrls" : [
      "http://example.com/images/foo1.png",
      "http://example.com/videos/foo2.mp4",
      "http://example.com/images/foo3.png",
],
```


INTEL SECTION: Analytic information generated by pipeline


```
"intel" : {
```


LANGUAGE ANALYSIS


```
"language" : "en",
"tokens" : [
      { "text" : "Lorem", "onset" : 0, "offset" : 5},
      { "text" : "ipsum", "onset" : 6, "offset" : 11},
      { "text" : "dolor", "onset" : 16, "offset" : 21},
],
"sentences" : [
      { "text" : "Lorem ipsum dolor sit amet, consectetur adipiscing elit", "onset" : 0, "offset" : 55 },
      { "text" : "Nulla mauris sem, blandit nec dignissim vitae, tincidunt id enim", "onset" : 57, "offset" : 121 },
],
```

ENTITIES


```
"entities" : {
	"persons" : [
		{
			"name" : "Angela Merkel",
			"count" : 2,
			"url" : "http://dbpedia.org/Angela_Merkel",
			"matches" : [
				{ "text" : "Angela Merkel", "onset" : 0, "offset" : 40},
				{ "text" : "Merkel", "onset" : 300, "offset" : 316},
			],
			"sentiments" : [
				{ "score" : 0.7, "eval" : "+", "onset" : 763, "offset" : 414 },
				{ "score" : 0.1, "eval" : "=", "onset" : 763, "offset" : 414 }
			]
		},
		{
			"name" : "Barack Obama",
			"count" : 1,
			"url" : "http://dbpedia.org/Barack_Obama",
			"matches" : [
				{ "text" : "Obama", "onset" : 200, "offset" : 231},
			]
		},
	],
	"locations" : [
		{
			"name" : "Berlin",
			"count" : 1,
			"url" : "http://dbpedia.org/Berlin",
			"matches" : [
				{ "text" : "Berlin", "onset" : 103, "offset" : 108},
			]
		},
	],
	"organisations" : [
		{
			"name" : "Siemens",
			"count" : 1,
			"url" : "http://dbpedia.org/Siemens",
```

```
                    "matches" : [
                            { "text" : "Siemens", "onset" : 208, "offset" : 215},
                    ],
                    "sentiments" : [
                            { "score" : "0.0", "eval" : "-", "onset" : 208,
"offset" : 300 },
                    ]
            },
      ]
},
```


SUMMARIES


```
"summary" : "Lorem vestibulum erat egestas",
```


RDF: Buffer for arbitrary RDF data in JSON-LD format.


```
"rdf" : [
  {
    "@id": "http://dbpedia.org/Barack_Obama",
    "@type": [ "http://xmlns.com/foaf/Person" ],
    "http://xmlns.com/foaf/knows": [
     {
        "@id": "http://dbpedia.org/Angela_Merkel"
      }
    ]
  },
  {
    "@id": "http://dbpedia.org/Angela_Merkel",
    "@type": [ "http://xmlns.com/foaf/Person" ]
  }
]
```


Table 1: CEP JSON Container

This approach allows for more efficient and robust execution: it does not require multiple writes and reads to repositories to store the results and obtain data. The JSON container is also highly compressible, so it can be efficiently sent over the network. It also ensures the behaviour is transactional: in case of an error during the pipeline run, there is no need to roll back multiple operations to prevent bad data from being stored. All results of the analysis are committed in a single write, in an “all or nothing” fashion.

Based on the type of the article being processed, it is routed through different paths of the pipeline (see Figure 10).

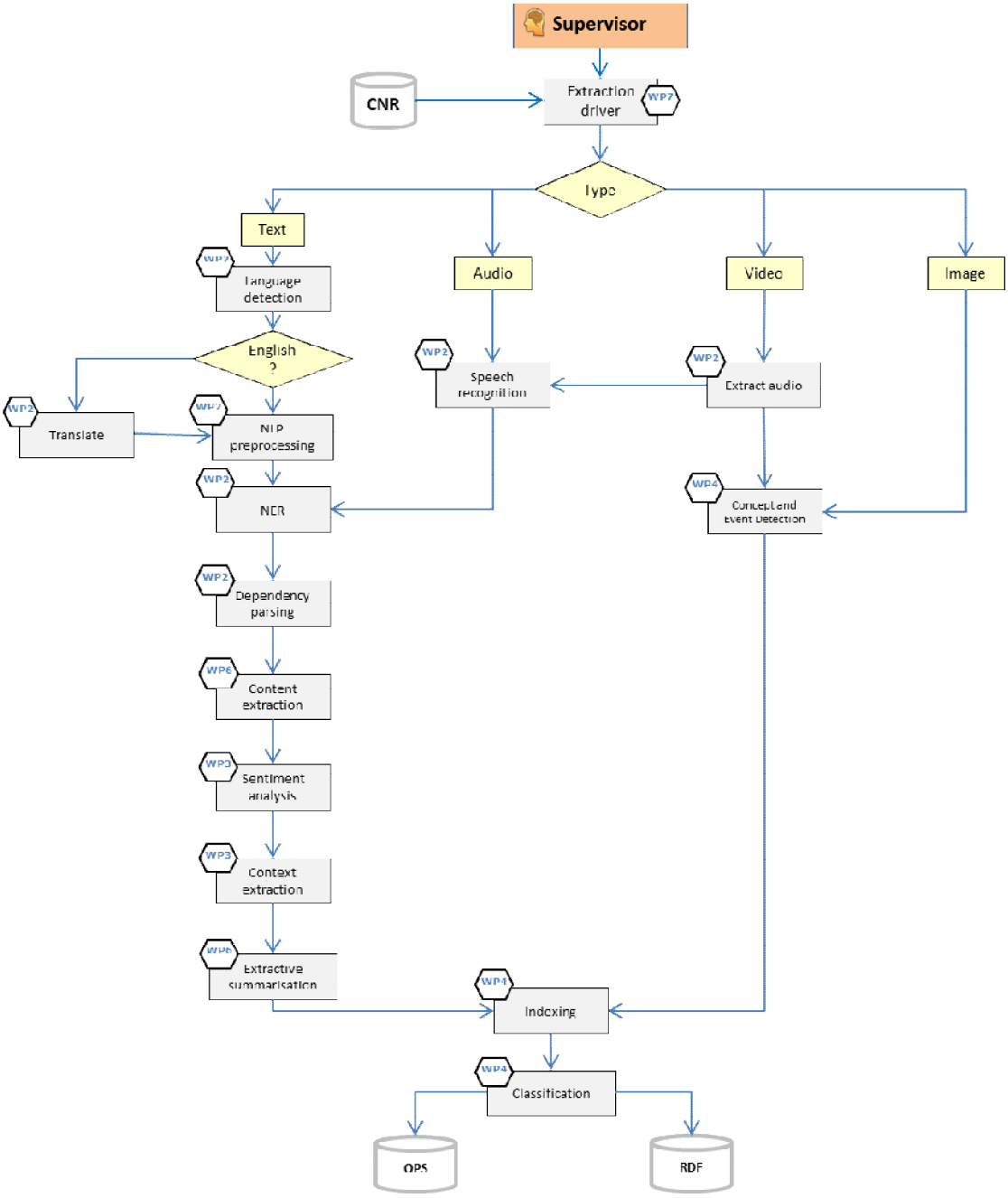

Figure 10: CEP execution

NLP Preprocessing Service [WP2]

Basic pre-processing of the text: tokenisation, stripping extra spaces, removing HTML and extraneous content (e.g. illegal UTF-8 content) if present.

Language detection Service [WP2]

Automatically determines the language of the source text.

Translation Service [WP2]

Non-English content is automatically translated to English for analysis purposes (see D7.1, section 3.1.6).

NER Service [WP2]

Recognises and marks some types of proper names (person names, geographical entities, company names etc.) in a text (see D7.1, section 3.1.1).

Dependency parsing Service [WP2]

Analyses texts in terms of the syntactic structure of sentences, i.e. grammatical relations between the different words of a sentence (see D7.1, section 3.1.2).

Content extraction Service [WP6]

Links detected entities and facts to ontologies in the RDF repository such as DBPedia or FreeBase, where possible, for enhanced semantic processing later on. (See D7.1, section 3.1.3)

Sentiment analysis Service [WP3]

Detects sentiments and their strengths in English text (see D7.1, section 3.2.2), and annotates the text with this polarity and sentiment information. Analysis will include detection of sentiments towards entities. Generated facts will be expressed as RDF.

Context extraction Service [WP3]

Extracts contextual features from a text (e.g. relevance of the article in social networks, mentions to topics) (see D7.1, section 3.2.1). Generated facts will be expressed as RDF and modelled according to a custom contextual feature ontology.

Extractive summary Service [WP6]

Generates a plain text extractive summary of an arbitrary text (see D7.1, section 3.5.2).

Indexing Service [WP4]

Indexes the analytical data into a custom structure for future searches (see D7.1, section 3.3.1).

Classification Service [WP4]

Determines the general category of an item, e.g. politics, economy, lifestyle, etc. (see D7.1, section 3.3.2).

Speech recognition (ASR) Service [WP2]

Converts human speech to text (see D7.1, section 3.1.4). Generated text is sent through rest of text pipeline.

Concept and event detection Service [WP4]

Detects concepts and events found in multimedia files (see D7.1, section 3.1.5).

At the end of the processing pipeline, the contents of the JSON container are persisted to repositories for later use by the client applications. There is a variety of data formats generated by the analysis services: RDF data is written to the RDF Repository, and structured and literal data is written to one or more databases in the OPS Repository.

4.2.2.4 Social Media Analysis pipeline (SMAP)

The Social Media Analysis pipeline (SMAP) (Figure 11) is a set of processes related to analysis of social network data stored in the MULTISENSOR repositories. It is executed periodically in the background by the Supervisor.

The SMAP pipeline analyses mainly Twitter and Facebook data to detect trends, map network graphs of contributors, and track topics and controversies over time (see D7.1, section 3.2.3).

Findings of the SMAP pipeline are expressed as RDF and written to the semantic repository. This information can later be used to service queries.

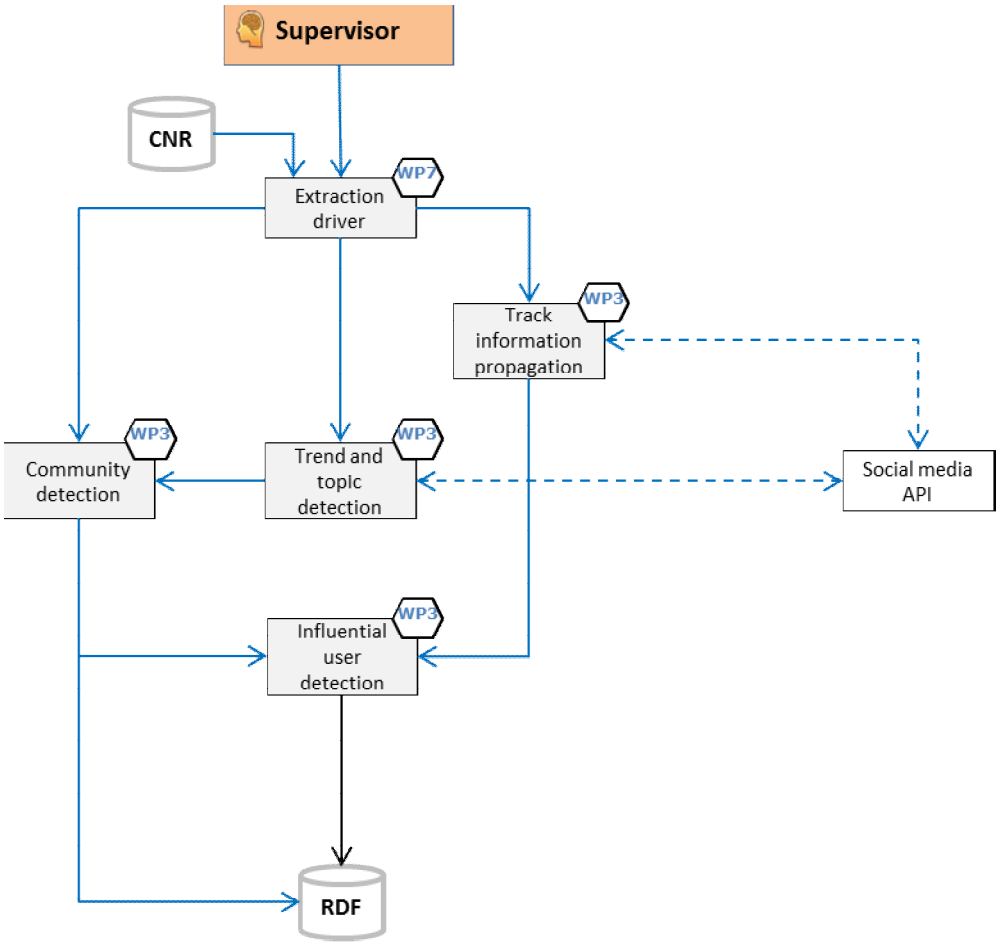

Figure 11: SMAP execution

4.2.2.5 Content alignment pipeline (CAP)

The Content Alignment pipeline (CAP) (Figure 12) is a background batch processes that analyses the existing data and performs reasoning to detect inconsistencies and contradictory facts, and generate implicit knowledge via inference (see D7.1, section 3.3.4).

The CAP is executed by the Supervisor at fixed intervals. It issues queries to the RDF and OPS repositories, and generates additional data from its findings, which is written back to the repositories.

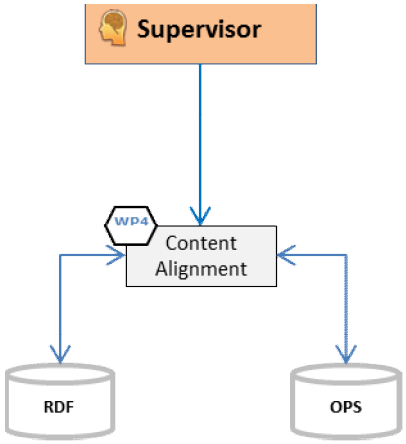

Figure 12: CAP execution

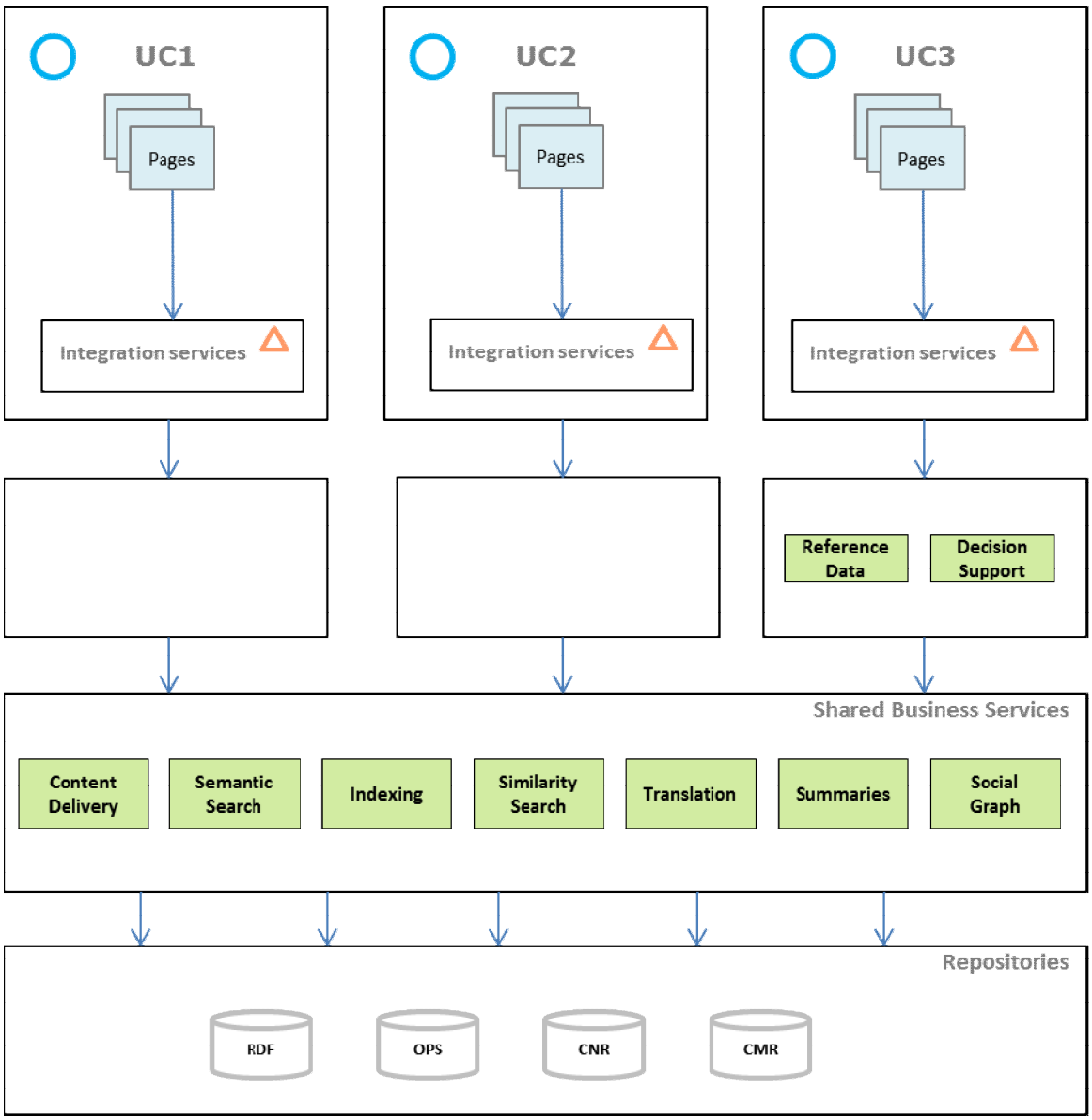

Figure 13: Online modality overview

4.2.3 Online modality

The online modality of the MULTISENSOR platform (Figure 13) is responsible for live interaction with users and the execution of live queries and the display of results.

For demonstration purposes, 3 separate scenarios will be built, to display the capabilities of the platform according to the requirements discussed in D8.2.

The scenarios will use a common set of services and repositories, and a scenario-specific presentation layer, to implement a UI tailored for each use case.

4.2.3.1 Shared business services

The core runtime capabilities of the MULTISENSOR platform are encapsulated as REST services shared across the different UC applications. Most of these services return raw data in JSON containers, and the output is formatted at the presentation layer by the different UC applications according to their specific needs and features.

Content delivery Service

Works as a broker to get the analytic information previously generated by the offline pipelines and stored across the repositories. Given a specific item, can be asked to return the summary, translations, sentiments information, etc. Can combine calls to several services or repositories to assemble a response.

Note that this service does barely any processing of its own apart from marshalling of the data into a JSON container: the data has already been processed as part of the analytic processes described in section 4.2.2.3.

Content delivery Service				
Methods				
`/news/{id}/raw`	GET	Get specified SIMMO	SIMMO ID	JSON
`/news/{id}/media/{mediaId}`	GET	Get specified multimedia item	SIMMO ID Media ID	Varies (MIME type of media)
`/news/{id}/summary`	GET	Get summary	SIMMO ID	JSON
`/news/{id}/entities`	GET	Get Entities	SIMMO ID	JSON
`/news/{id}/sentiments`	GET	Get sentiments information	SIMMO ID	JSON
`/news/{id}/trans/{lang}`	GET	Get translation	SIMMO ID Target language ID	JSON
`/news/{id}/all`	GET	Get all available information	SIMMO ID	JSON

Machine translation service

Provides on-demand, synchronous translation of arbitrary text to a target language.

Machine translation service
Methods

/translate	POST	Translates content	Source language Target language Text	JSON

Semantic search Service

The Semantic Search service is a multi-purpose, multi-modal entry point to the core MULTISENSOR search capabilities.

It will support several modes of operation:

- **FTS Mode**: classic Full Text Search mode of the entire repository, supporting filters, facets and field queries. Will forward queries to the ElasticSearch instance hosting the CNR.
- **Semantic Mode**: search on top of the RDF repository. Will leverage inference and ontologies to capture user intent and return relevant results considering synonyms, equivalent concepts, hierarchical concepts, etc. Will dynamically construct SPARQL queries to be run against the RDF repository. Will work in close cooperation with the Indexing Service (see below), which will provide capabilities such as automated clustering of results and categorisation.
- **Hybrid mode**: a combination of FTS and semantic, doing a semantic pre-processing and enrichment of the search query (via queries to the RDF knowledge base) to assemble an ElasticSearch query for extra accuracy and relevancy (experimental).

All modes of operation will support filtering, paging and optionally exclusion of blacklisted results.

Semantic Search Service				
Methods				
/search/fts	POST	Search the repository	Search query (ElasticSearch JSON object)	JSON
/search/rdf	POST	Search the repository	Search query	JSON
/search/hybrid	POST	TBD	TBD	TBD

Indexing service

The Indexing Service will use a multimodal indexing structure (described in D7.1, section 3.3.1) to efficiently retrieve generated analytical information from the OPS and RDF repositories.

The different search modalities will use this service to retrieve information such as automatically assigned categories and clustering of results.

Similarity search Service

Given an image, returns similar or related images, i.e. other images sharing contextual features with the original (e.g. logos, brand names, locations). Will work in cooperation with the Indexing service to retrieve multimedia indexing information (see D7.1, section 3.3.1).

Summarisation service

Provides on-demand abstractive summaries of arbitrary texts or bundles of texts (see D7.1, section 3.5.3)[9]. For extractive summaries, they're pre-generated during the analysis pipeline, so they can be retrieved directly via the Content Delivery service.

Summarisation Service				
Methods				
`/summarise`	POST	Generate summary for specified text fragment	Raw input text	JSON

Social graph Service

Provides information about people or entities in the networks analysed, i.e. details about a specific journalist's influence network, connections, influence, relationships, etc. Will leverage information collected and generated during the Social Media Analysis offline process described in section 4.2.2.4.

Social Graph Service				
Methods				
`/people/{name}`	GET	Get all information available about NAME	Person name, or social media handle, or outlet name	JSON
`/people/{name}/network`	GET	Get influence network for NAME	Person name, or social media handle, or outlet name	JSON

4.2.3.2 UC1 – Journalism use case

[Functional description and high-level requirements for UC1 are discussed in D8.2]

The general theme of the UC1 application is providing journalists with a framework that enables processing and understanding of large amounts of news in a quick and agile way, allowing users to "zoom in" and "zoom out" on topics of interest and explore the available data using several dimensions.

Functionally, the UC1 application (Figure 14) is a rich search engine and visualisation interface on top of the MULTISENSOR platform. It allows the user to do an initial free-form search, and then drill down on the results by filtering using sub-searches and dimension filters (e.g. language, date, country…).

A "dossier" provides the user with a virtual container to store the items relevant to the issue being investigated for later review.

[9] Abstractive summarisation is highly experimental and it is a complex process involving intensive computation, which might make it unsuitable for synchronous use; if performance is found to be too low, it will be implemented as an asynchronous service, delivering results through a queue subsystem.

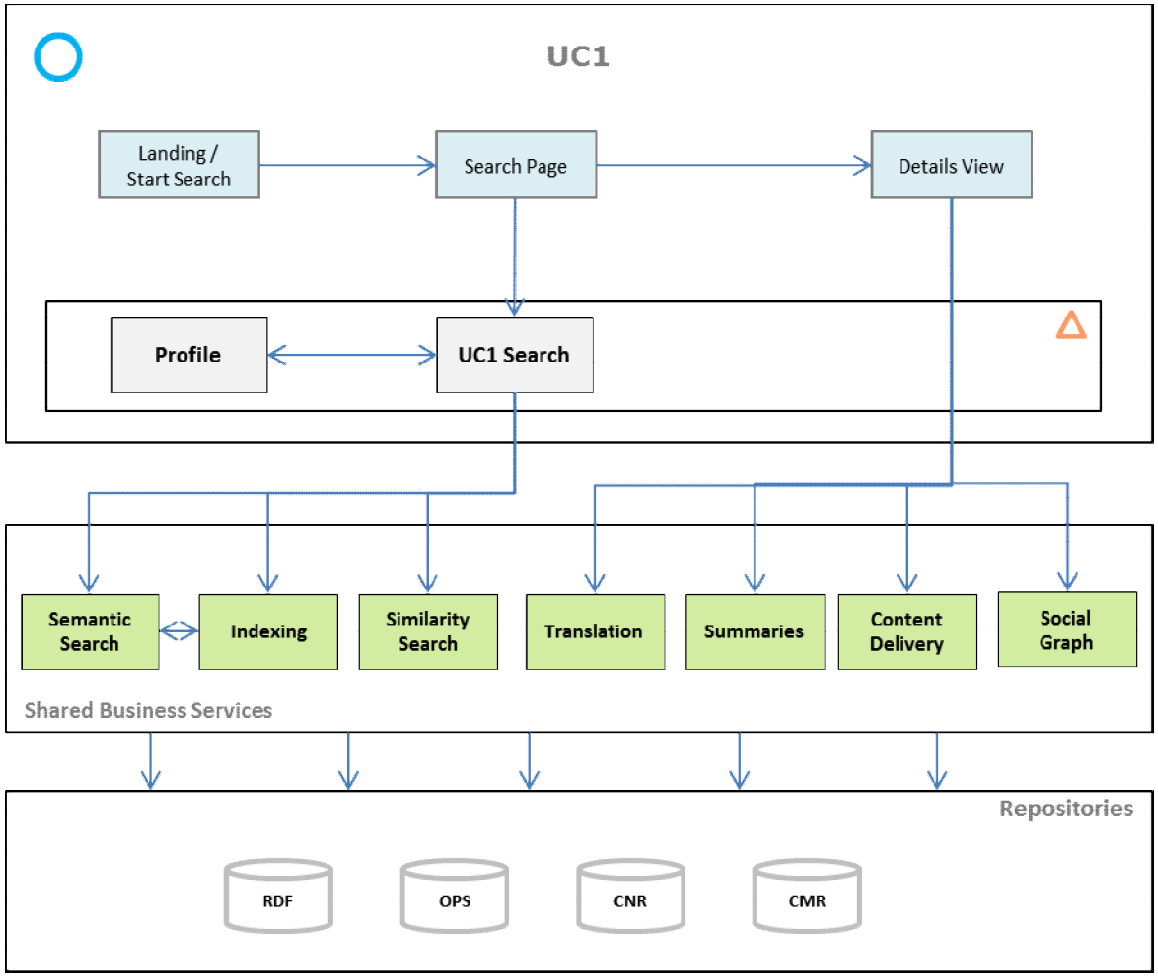

Figure 14: UC1 application diagram

UC1 Profile service

Manages authentication to provide user profiles. Popular authentication methods such as Google Accounts will be supported for frictionless authentication into the system.

Users have “dossiers”: a dossier contains the search terms, plus all items that have been flagged by the user as relevant to that dossier, and miscellaneous preferences, such as a blacklist of items flagged irrelevant, and filter configuration.

UC1 Search service

A thin wrapper around the shared search services: implements extra functionalities like automatic filtering of irrelevant items, applying profile preferences to the search, paging and sorting, etc.

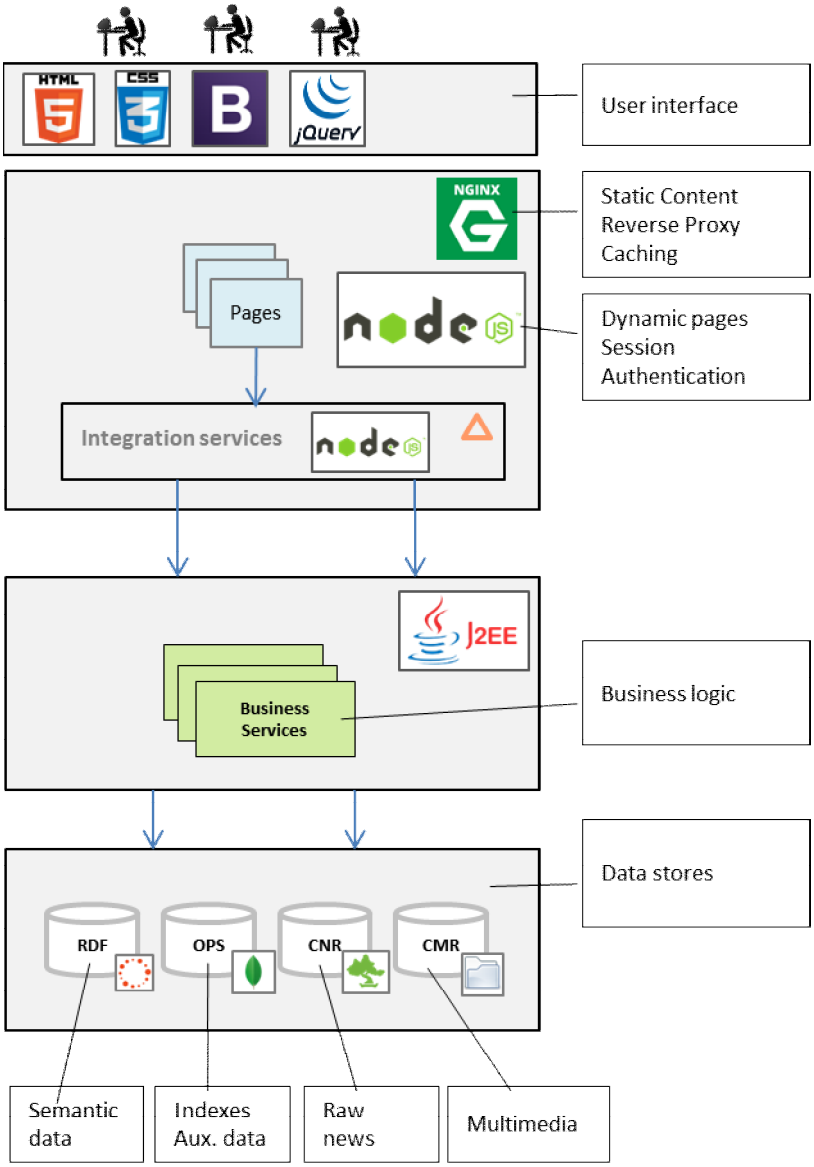

Figure 15: UC1 technical implementation

Implementation

The UC1 application is built as a Node.js web application using the Locomotive MVC framework[10]. Pages and client-side AJAX functionality are built using HTML5, CSS3 and javascript. The UC1 technical implementation is illustrated in Figure 15.

4.2.3.3 UC2 – Media Monitoring use case

[Functional description and high-level requirements for UC2 are discussed in D8.2]

The UC2 application (Figure 16) is focused on replication of a media monitoring professional's workflow for execution of an analysis. The process involves the following steps:

- Defining the sources and search terms along with additional parameters;
- Curation and validation of the results;

[10] See http://locomotivejs.org/

- Classification of results;
- Multi-dimensional analysis of the final data set.

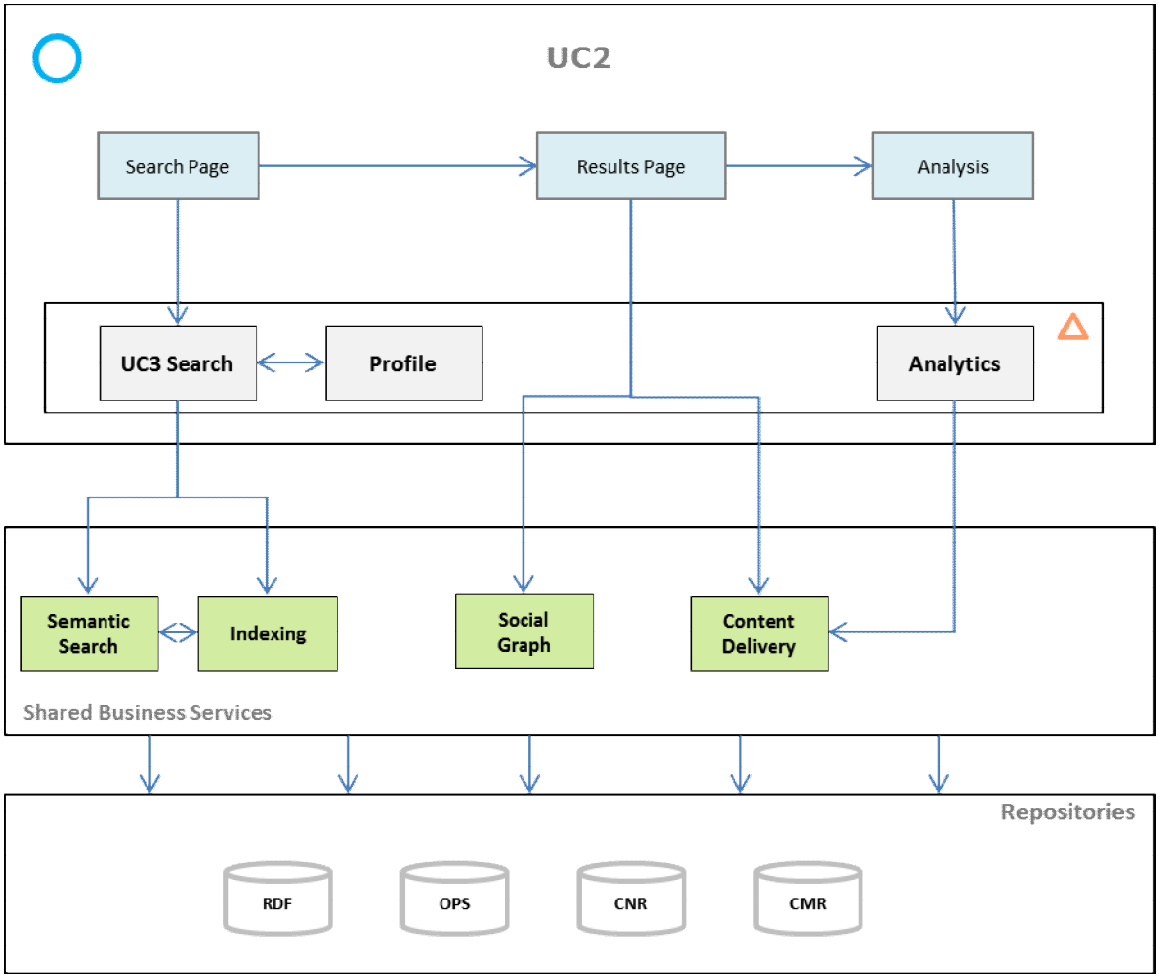

Figure 16: UC2 Application diagram

The application will make use of the shared business services of MULTISENSOR to execute the searches, and leverage the automated clustering and classification pipelines for automated classification of the search results. It will also use the Content Delivery and Social Graph services to retrieve details about the news items and author information.

UC2 Profile service

Implements "search profiles", i.e. a set of configurations replicating a real-world scenario for an analysis. The profile stores preferences such as blacklisted results, preferred synonyms, etc.

UC2 Search service

A wrapper around the core Semantic Search service. Applies filtering based on profile preferences and any auxiliary features required (e.g. caching of results).

UC2 Analytics service

Implements smart visualisations of a dataset. Specific analysis descriptions and formats are not yet fully defined.

Implementation

The UC2 application will enhance the existing NewsRadar application from pressrelations: rather than build an application from scratch, integration interfaces will be built to leverage functionality provided by the MULTISENSOR services from the NewsRadar application. The UC2 technical implementation is depicted in Figure 17.

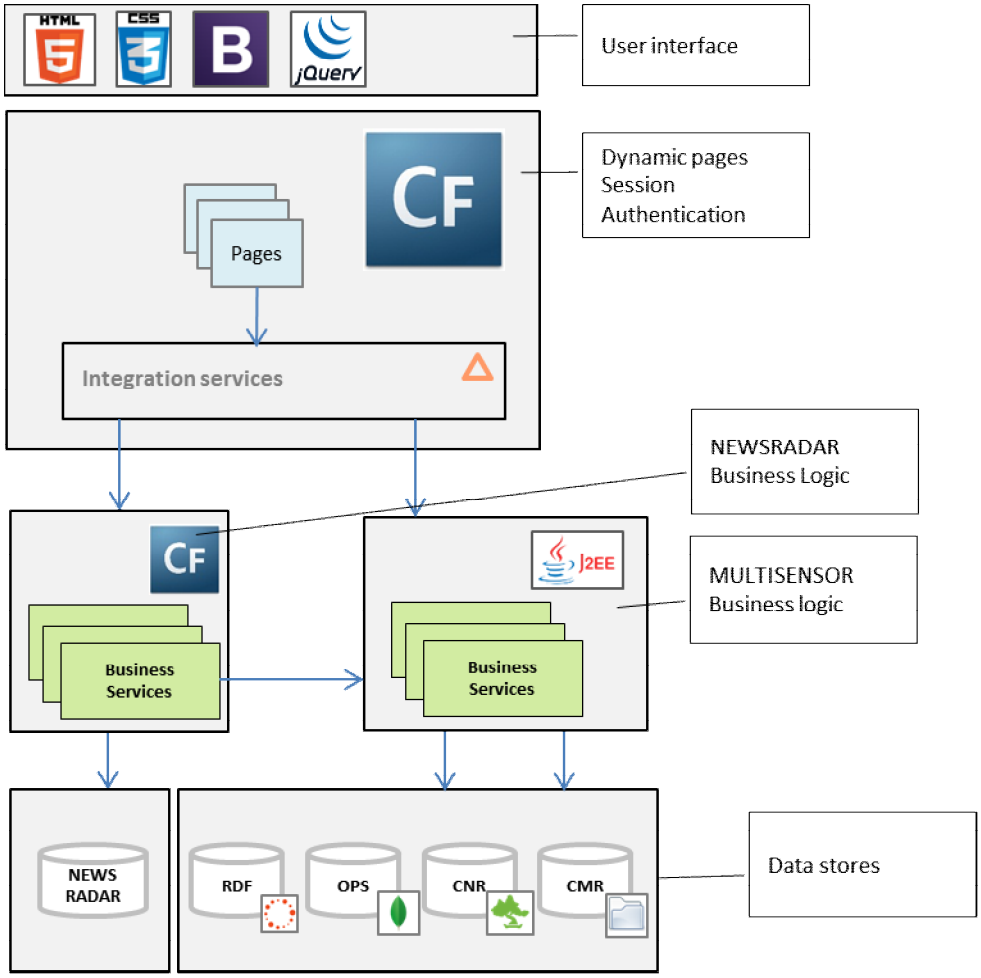

Figure 17: UC2 Technical Implementation

4.2.3.4 UC3 – SME internationalisation use case

[Functional description and high-level requirements for UC3 are discussed in D8.2]

Functionally, the UC3 application (Figure 18) provides an exploratory interface, enabling users to navigate information about countries and market sectors, and incrementally building a scenario for a decision making process regarding expansion of business to a specific EU country.

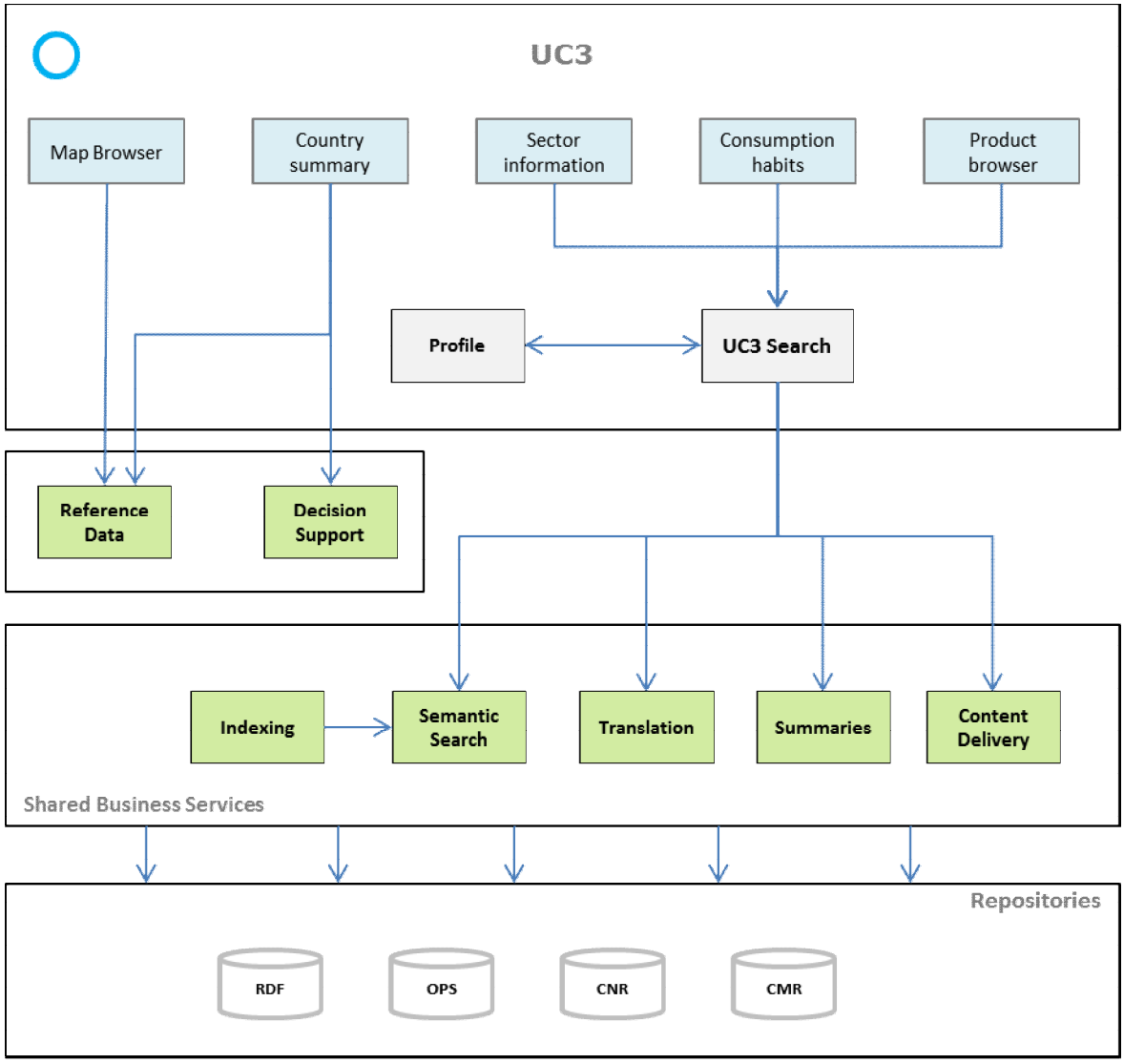

Figure 18: UC3 Application diagram

UC3 Profile service

Manages authentication to provide user profiles. Popular authentication methods such as Google Accounts will be supported for frictionless authentication into the system.

UC3 Reference data service

An API to access reference data for display purposes. Reference data includes demographic, statistical and economic data about EU countries. This information will be obtained from a variety of sources: targeted crawling of reference data sites (see section 4.2.2.2), Ontotext's FactForge aggregated dataset[11], and any static datasets required.

UC3 Decision support Service

Implements an API to expose the functionalities of the decision support component built on top of the semantic repository (see D7.1, section 3.4.3). In the context of the UC3

[11] See http://factforge.net/

application, it will be used to assist the user in evaluating business expansion to a designated country, based on facts present in the RDF repository.

Implementation

The implementation (Figure 19) will be conceptually identical to the UC1 web application and deployed on the same infrastructure.

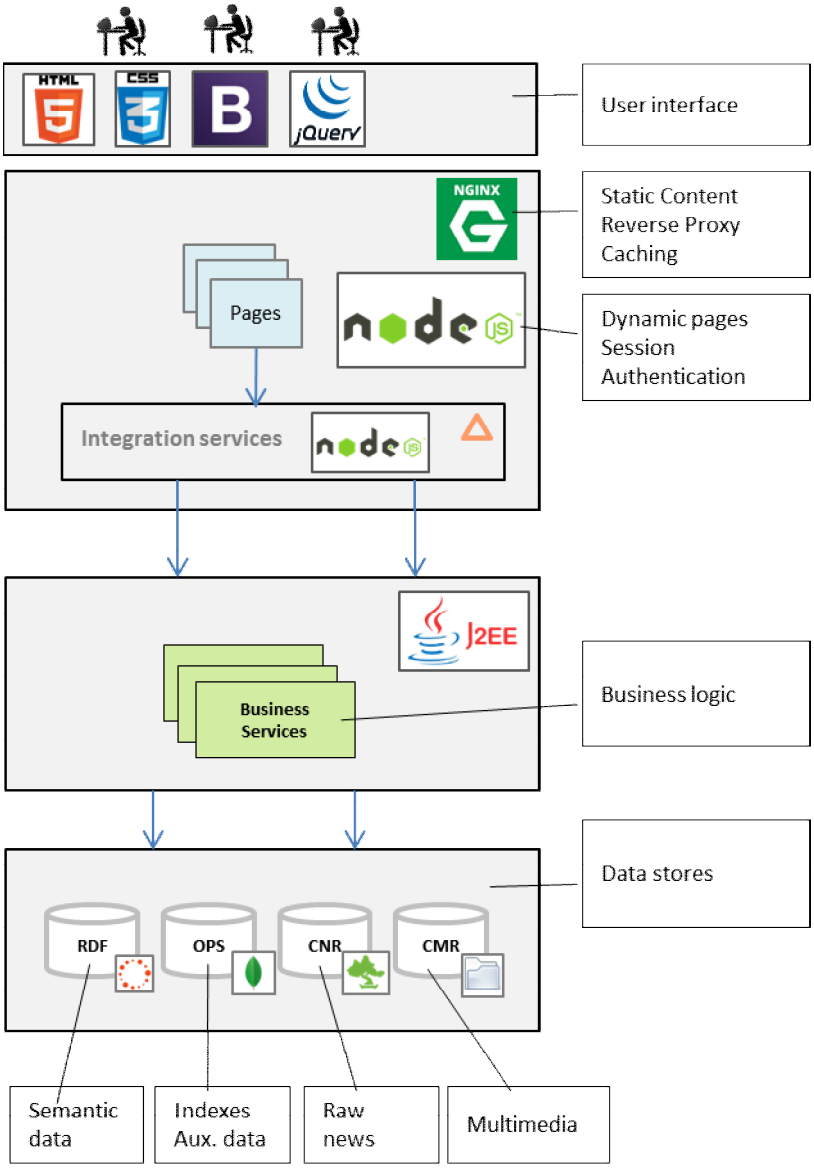

Figure 19: UC3 technical implementation

4.2.4 Data stores

4.2.4.1 Central News Repository (CNR)

The CNR stores all harvested news. The crawler stores news items (news articles, social media posts, etc.) as they become available, along with metadata (source, date, country, etc.). Unprocessed news can then be pulled by the analytic pipelines for processing.

The CNR is also used for FTS searches over the dataset, for keyword-based lookups over news titles and bodies.

Implementation: the CNR is built on top of **Elasticsearch**[12], a next-generation, distributed real time search engine built on top of the classic Lucene search technology. Elasticsearch scales to very big datasets with very high performance and offers rich querying and analysis JSON-based APIs. It provides additional features such as automatic facet support and data percolation, which match well with the MULTISENSOR requirements.

4.2.4.2 Central Media Repository (CMR)

The CMR stores source multimedia content (video, images and audio) collected by the harvester. Multimedia content is then picked by the analytic pipelines for processing.

Implementation: the CMR is built as a simple filesystem. Multimedia items are retrieved and dumped in a folder structure. The tree is then exposed over HTTP for use by the rest of the platform.

4.2.4.3 Semantic Repository (RDF)

The RDF repository holds all data in semantic format for the platform. This includes the ontologies and datasets used to model the data, as well as semantic annotations on the harvested SIMMOs. The RDF repository is searchable via SPARQL.

See D7.1, section 3.4 for discussion of the configuration and deployment of the MULTISENSOR semantic infrastructure.

Implementation: implemented as an instance of **GraphDB** (formerly known as OWLIM), developed by Ontotext[13]. GraphDB provides a solid, scalable platform for high performance semantic storage with advanced reasoning capabilities.

4.2.4.4 Operations Repository (OPS)

The OPS repository provides fast, read/write structured data storage for any systems in the platform that require it.

Implementation: implemented as an instance of **MongoDB**[14]. It provides fast, scalable document-oriented storage that suits well for the purposes of MULTISENSOR, given that there are no hard transactional requirements. MongoDB can be queried using a JSON API.

4.3 Standards and conventions

The distributed, multi-disciplinary, multi-team nature of the MULTISENSOR project requires a strong emphasis on standards enforcement and shared semantics in order to achieve reliable exchange of information and effective processing of data.

This section provides the general guidelines that will be followed to implement the vision of a distributed service and information architecture.

4.3.1 Exchange formats

As a general rule, the preferred format for data exchange and representation in the MULTISENSOR platform will be JSON[15] for structured data and RDF[16] (in JSON-LD[17], Turtle[18], N-Triples[19] or RDF-XML[20] transport) for semantic data.

---

[12] See **http://www.elasticsearch.org/**

[13] See **http://www.ontotext.com/products/ontotext-graphdb/**

[14] See **https://www.mongodb.org/**

XML will be used when required by specific tools or systems, or when there is a clear advantage in doing so (i.e. better performance, pre-existing optimised tool or specific domain dialect, etc.).

4.3.2 Encodings

The encoding for all textual data in the platform is UTF-8. All documents, subsystems and tools will be configured to accept and emit UTF-8 and ensure correct handling of multi-lingual content.

4.3.3 Compression

For systems in the platform that use it, the compression standard will be GZIP[21], which is widely supported, is cross-platform and offers reasonable performance and compression ratio.

4.3.4 Namespaces

For formats that allow/require namespaces, the canonical namespace root for document data is **http://multisensorproject.eu**. Definitions of entities and their members should be nested under the canonical root.

[15] See **http://www.json.org/** for formal description of the JSON standard.

[16] See **http://www.w3.org/RDF/** for an introduction to the RDF standard.

[17] See **http://json-ld.org/** for information on the JSON-LD standard.

[18] See **http://www.w3.org/TeamSubmission/turtle/**

[19] See **http://www.w3.org/TR/rdf-testcases/#ntriples**

[20] See **http://www.w3.org/TR/rdf-syntax-grammar/**

[21] See **http://www.gnu.org/software/gzip/**

5 INFRASTRUCTURE

The MULTISENSOR Platform is expected to deal with large amounts of data and require considerable resources to process store and search the dataset. The amount of required horsepower will increase as the project progresses and algorithms become more complex and the main data corpus grows larger.

In order to accommodate the increasing workload and support painless scaling, the main MULTISENSOR farm will be deployed in a cloud environment. Using a cloud service will allow the project to grow as needed without big upfront costs in terms of hardware and IT work. Cloud providers also typically have high speed network links, which make communication between servers and to/from the internet very efficient.

The MULTISENSOR architecture is designed with scalability in mind: one of the design goals is to avoid having a single chokepoint that might compromise growth of the platform as the dataset grows big and the analytic processes and heavy duty runtime queries become demanding on hardware resources. The Reference MULTISENSOR Cloud infrastructure is depicted in Figure 20.

All the elements and technologies used for implementation of the architecture can be scaled, either horizontally (e.g. via clustering or sharding) or vertically[22] (by relocating the service to a virtual server with higher specs, or assigning more resources).

For example, ElasticSearch and MongoDB support sharding: search indices can be partitioned across multiple servers, with a broker subsystem that automatically redirects requests to the right server.

OWLIM can be clustered: several servers running OWLIM can be placed behind a load balancer, and requests are distributed among them. Data updates are transparently propagated throughout the cluster for synchronisation. Hadoop itself is built on the concept of horizontal scalability: its value proposition is splitting a workload into small, parallelizable tasks and running them over a small fleet of computers (frequently inexpensive, commodity servers, as opposed to a high-end massive processing servers).

The initial farm layout foresees the following:

- **Web frontend**: running Nginx and Node.js for hosting UC1 and UC3 web apps.
- **Grinder server**: for running the analytic pipelines (Jetty + J2EE services). This will run heavy duty processes such as multimedia analysis, dependency parsing and sentiment analysis.
- A pool of **Hadoop worker nodes** will be available for asynchronous bulk processing of data.
- **Business services server**: for running the online business services (Jetty + J2EE services).
- **LT server**: for machine translation and Automatic Speech Recognition work (running on LINGUATEC premises). This is actually a proxy to a farm of servers doing the work and maintained separately.

[22] See **http://en.wikipedia.org/wiki/Scalability#Horizontal_and_vertical_scaling** for discussion on horizontal and vertical scaling

- **OWLIM server**: for running OWLIM (running on ONTOTEXT premises).
- **OPS/CNR**: for hosting the OPS (MongoDB) and the CNR Repositories (ElasticSearch).
- **CRAWLER1:** for running the Supervisor and the PR Collector (Node.js).
- **CRAWLER2**: for running the Targeted Crawler (J2EE, Hadoop).

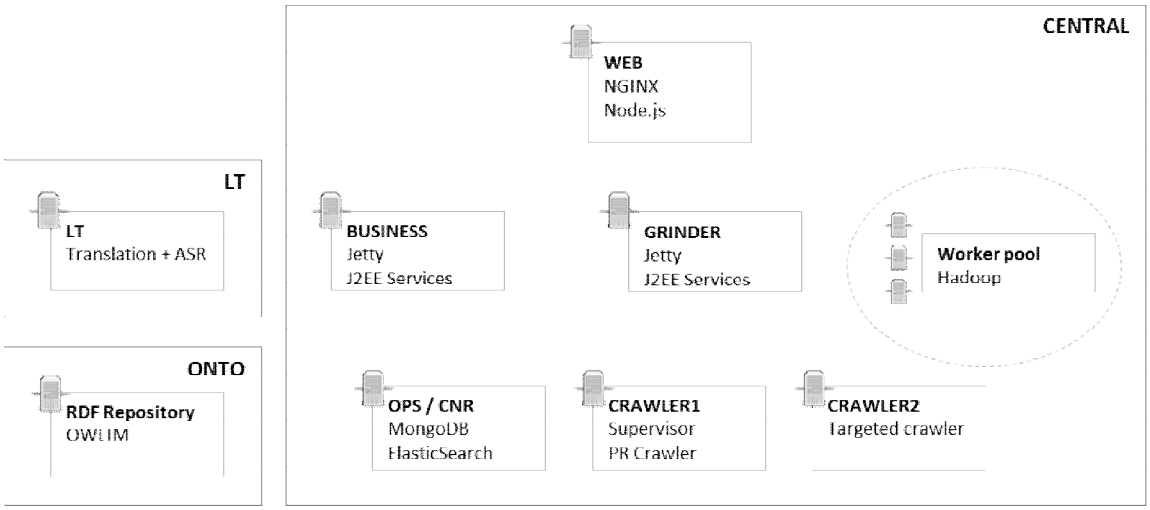

Figure 20: Reference MULTISENSOR Cloud infrastructure

6 CONCLUSION

The architecture specification presented in this document provides a detailed view of the conceptual and technical vision of the future MULTISENSOR platform. It consolidates the work done by the Consortium into the initial conceptualisation, roadmap (D7.1), gathering of requirements (D8.2) and day-to-day discussion into a concrete vision that will greatly assist partners in coordination of their work and integration of the different components, providing a blueprint and a path towards the complete system.

The proposed service-oriented architecture with low coupling between components allows for easy separation of work and the enforcing of open standards and formats allows for interoperability. Encapsulation of components in services will also increase reusability in future integrations of the technology into other systems.

Although this architecture is design following the current user (D8.2) and technical requirements it might need to be updated and/or adapted in case additional user requirements arise after the evaluation of the 1st prototype (M20). Such updates will be reported in the upcoming WP7 deliverables.

7 REFERENCES

Fowler, M. 2002. “Patterns of Enterprise Application Architecture”. Addison-Wesley.