Project Number 780251

D2.2 Text Modelling Extension

Version 1.1
27 December 2018
Final

Public Distribution

Edge Hill University

Project Partners: **Alpha Bank**, **ATB**, **Centrum Wiskunde & Informatica**, **CLMS**, **Edge Hill University**, **GMV**, **Nea Odos**, **The Open Group**, **University of L'Aquila**, **University of Namur**, **University of York**, **Volkswagen**

Project Partner Contact Information

Alpha Bank Vasilis Kapordelis 40 Stadiou Street 102 52 Athens Greece Tel: +30 210 517 5974 E-mail: vasileios.kapordelis@alpha.gr	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany Tel: +49 421 22092 0 E-mail: scholze@atb-bremen.de
Centrum Wiskunde & Informatica Tijs van der Storm Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 9333 E-mail: storm@cwi.nl	**CLMS** Antonis Mygiakis Mavrommataion 39 104 34 Athens Greece Tel: +30 210 619 9058 E-mail: a.mygiakis@clmsuk.com
Edge Hill University Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk	**GMV Aerospace and Defence** Almudena Sánchez González Calle Isaac Newton 11 28760 Tres Cantos Spain Tel: +34 91 807 2100 E-mail: asanchez@gmv.com
Nea Odos Charalampos Daskalakis Themistocleous 87 106 83 Athens Greece Tel: +30 210 344 7300 E-mail: cdaskalakis@neaodos.gr	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of L'Aquila Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L'Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it	**University of Namur** Anthony Cleve Rue de Bruxelles 61 5000 Namur Belgium Tel: +32 8 172 4963 E-mail: anthony.cleve@unamur.be
University of York Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk	**Volkswagen** Behrang Monajemi Berliner Ring 2 38440 Wolfsburg Germany Tel: +49 5361 9-994313 E-mail: behrang.monajemi@volkswagen.de

Document Control

Version	Status	Date
0.1	Document outline	24th September 2018
0.2	First draft	10th October 2018
0.7	First full draft	28th November 2018
0.8	Further editing draft	8th December 2018
1.0	Review version	10th December 2018
1.1	Final QA updates	27th December 2018

Table of Contents

Executive Summary

Text modelling is used to identify data-types and their relations in text. It is a basis for text processing components and is essential step in text platforms such as UIMA. We present a type system required to support TyphonML, i.e. a new modelling language to support the design of hybrid polystores. We also evaluate our type system by developing text mining pipelines that build upon the type system in order to extract various type of information from large-scale collections, such as base form of words, i.e. stems, and polarity of text.

Firstly, we review previous literature related to different types of text annotations, i.e. standoff and inline, various platforms supporting annotation, i.e. Tipster, General Architecture for Text Engineering (Gate), Ellogon, Heart of Gold(HoG) and Unstructured Information Management applications (UIMA), and type-systems. Consequently, we found from the literature, after a comparison on standoff and inline annotations that standoff annotation are more beneficial for our purposes in this project. As for the platform, we chose UIMA due to its multiple benefits.

We categorised data-types according to the type of textual unit, i.e. Document level, Paragraph level, Sentence level, Phrase level, and Token level, and also according to the type of information that they annotate, i.e. syntactic or semantic. Extendibility and flexibility of our type system is demonstrated with some use-case examples specific for different project partners requirements.

Lastly, we demonstrate how the proposed Typhon text data type system can be used for the development of text processing pipelines in the real-world. We develop different sentiment analysis UIMA text processing pipelines and evaluate the performance against other several publicly available datasets. Consequently, the results show that our type system can aid rapid development of sentiment analysis pipelines wherein text processing components can be freely combined using shared data types.

1 Introduction

Most search and analytics engines, such as ElasticSearch[1], allow searching text efficiently, however they have limited search capabilities other than plain text, for instance, keyword-based search. To allow more complex queries, such as faceted or semantic search, annotations, that are automatically extracted from text using text processing pipelines, can be added in existing search engines as facets. For example, consider a conventional keyword-based search that uses the query "Apple". Considering that this query is ambiguous, a keyword-based search will return documents referring both to "Apple" the IT company and documents referring to "Apple" the fruit. However, using text mining we can automatically identify the semantic roles of keywords, i.e. Organisation vs. Fruit, and then use semantic roles to enable efficient faceted search. In a faceted search scenario, a user is able to include the facet "Organisation" to the keyword "Apple" in order to retrieve documents that are relevant to "Apple" the IT company only and not to "Apple" the fruit.

Text modelling identifies data types for storing text and the relationships between them [12]. It helps to design text mining tasks and is a gateway to create text processing pipelines. Additionally, it allows developing complex text processing components helpful in facet searches, e.g. search for the sense of bank that relates to rivers rather than financial institutions, identifying new information, e.g. categories or classes, modelling topics and classifying text. Text modelling supports data abstraction [30]. Text can be combined or replaced with other text data types, if required. For example, words can be replaced with their corresponding lemma forms, which leads to better indexing and more efficient search [44]. For instance, a search query for the lemma "computer" will retrieve documents that contain the words "computing", "computers" and "computer", since these words have the same lemma, i.e. compute. Moreover, text modelling can improve the quality of semantic retrieval. For example, when searching for "iPad", the query term will be semantically associated with the organisation Apple [74]. Therefore, in a sentiment analysis application for example, we link "iPad" and "iPhone" as products of the same company. If "iPad" has been reviewed positively, "iPhone" may too. Furthermore, semantic disambiguation can subsequently improve machine translation, e.g. "Apple" in the above example refers to the company and not the fruit.

Text data types can be categorised according to the type of the textual unit that they annotate, such as Document level, Paragraph level, Sentence level, Phrase level and Token level. They can also be categorised according to the type of information that they annotate, for instance syntactic or semantic. Syntactic data-types include Part of Speech (PoS) tags, syntactic dependencies and lemmas. Semantic data-types can be named entities, document polarities, facts, events, domain or negations.

Textual data types can also be divided into complex and primitive ones. Complex data types contain attributes, i.e. references to other data types, which can in turn be either complex or primitive. An attribute is a property that helps to describe an instance of the type. A primitive data type can only contain one value which is either a String, Boolean (true or false) or a numerical value. For example, a phrase is a complex data type that consists of the following attributes: begin offset, end offset and a list of words. The begin and end offsets are primitive data types, in particular integers, which denote the offsets of a word\phrase within the document they occur. The list of words is a complex data type which references the constituent words of the phrase.

There are many Natural Language Processing (NLP) tools for designing and developing data type systems, i.e. collection of data types including the relationships between them. Examples of these tools include Tipster, the General Architecture for Text Engineering (GATE), Ellogon, the Heart of Gold (HoG) and IBM's Unstructured Information Management Architecture (UIMA). UIMA has many advantages over other NLP frameworks. Bank and Schierle [4] argue that UIMA is the most evolved and comprehensive architecture available. They compared UIMA against other frameworks and found that it is better. UIMA is Apache li-

[1] www.elastic.co/products/elasticsearch

censed and is standardised. It can integrate different standard file formats like XMI, Ecore, XML, OCL or BPEL. Furthermore, it provides parallelisation and distribution of processing resources [4]. It is also the only architecture that provides resource management and analysis-aware parameter handling [4].

UIMA allows users to define a type system which provides clear metadata about expected data types: inputs, e.g. tokens and sentences, and outputs, i.e. emotion or polarity labels [4]. The type system can specify data structures useful for the processing and sharing of information. Type systems allow data types to pass flexibly between components of an NLP system [4]. UIMA support stand-off annotations, meaning that they do not interfere or change the input text [79]. Stand-off annotations have several advantages over inline annotations. They identify annotations from overlapping hierarchies [31]. For example, when identifying paragraphs, sentences and tokens which offsets overlap with each other. In UIMA, they are annotated separately from the original file. Another example is overlapping semantic and syntactic types. They can be dealt with separately and independently. Finally, stand-off annotations can handle text which is read-only or secured [79].

In this deliverable, we present a type system of data types related to text processing and inspired by the analysis of textual data contributed by Typhon use case partners. The type system is part of TyphonML, a new modelling language, developed as part of Typhon, to support the design of hybrid polystores. Using this type system and the corresponding text processing workflows, textual data in the polystore will be processed and enriched with extra information that allow for sophisticated search capabilities. In this deliverable, we also evaluate our type system by building example sentiment analysis workflows on top of it. The workflows extract various types of information from large-scale collections of text, such as stems, the base form of words, and the polarity of sentiment expressed in text. In the following subsections, we will present an overview for the deliverable, the intentions and the outcomes.

1.1 Overview

The deliverable consists of nine sections. In section 2, we review the state-of-art in text modelling. This includes literature related to text annotations, platforms that support text annotations and existing type systems. In section 3, we present the type system that was developed to address the general text processing requirements of Typhon use case partners. Section 4 emphasises the ease of extensibility of the type system, in case of further text processing needs, and explains that this work can be used as a pilot for adding support for other types of data, such as images and video, to Typhon polystores. In section 5, we demonstrate how (a part of) the type system is extended to cover text processing needs that are specific to each Typhon use case partner. Section 6 presents an application of the type system for sentiment analysis and our experiments on large collections of text. The risks and shortcomings are illustrated in section 7. Typhon text processing requirements are revisited in section 8 followed by the conclusion, in section 9.

1.2 Intentions

The objective of this deliverable is to present the findings and conclusions of our investigation of text-related data types to be integrated into TyphonML. The state-of-the-art of NLP architectures is explored, explaining their advantages, disadvantages and possible consequences and risks associated with their use.

The text processing requirements of Typhon use case partners were analysed and a type system that addressed them has been developed. We collaborated with the University of L'Aquila (UDA) to ensure the data types are consistent with TyphonML. We have also started working with the Institut für angewandte Systemtechnik Bremen GmbH (ATB) on how text processing pipelines will be deployed and executed within the hybrid polystore.

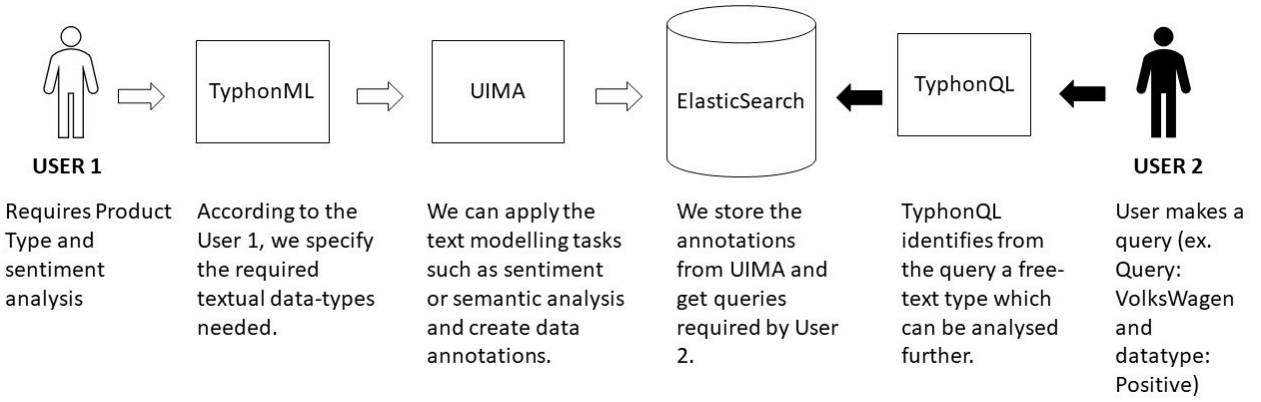

Figure 1: Text Modelling and Type System in the Context of Typhon Polystores

To verify that the newly developed type system is fit for our purpose, we developed a number of text processing workflows for sentiment analysis, we executed multiple experiments and evaluated them using state-of-the-art metrics. Consequently, we achieved competitive results in comparison to previously published research.

1.3 Outcomes

The outcome of this deliverable is a type system for text processing to be integrated into TyphonML. The type system contains both generic types and specialised ones for particular tasks, such as rule and pattern matching and machine learning classification.

Figure 1 illustrates an example of how text modelling and the newly developed type system fit into the Typhon project. In this example, there are two types of users, namely *User 1*, a Typhon database owner responsible for creating and maintaining a database and *User 2*, a Typhon user who wishes to analyse or search for information stored in the textual fields (e.g. customer review, product description) of the database. User 1 defines the different types of information found in the textual fields of the database while User 2 forms faceted queries that are compatible with the data types defined by User 1. More specifically, User 1 uses the TyphonML to define a) appropriate text data types from the Typhon text data type system that will be used to annotate the textual fields of the database and b) text processing pipelines (implemented using the UIMA framework) that analyse the textual fields of the database and generate the required text data types. The produced text data types are subsequently stored and indexed in ElasticSearch. User 2 uses TyphonQL to develop faceted queries where query-keywords are assigned to facets corresponding to the text data types indexed in ElasticSearch. For example, the following faceted query: “volkswagen suv tiguan”, year: 2018, sentiment: “Negative”, assigns the facets *year* and *sentiment* to the query keywords *2018* and *Negative*, respectively. The query is dealt with by TyphonQL, which accesses ElasticSearch to retrieve matching entries and shows them to User 2.

It should further be noted that the UIMA text processing pipelines, that populate the ElasticSearch back-ends with text data types, are batch-processing rather than on-line processing tools. In practice this means that the text processing pipelines are executed *offline* and not during query execution time. Text search capabilities (e.g. faceted queries) will be made available to User 2 only after the UIMA pipelines have finished processing the textual fields of the database.

2 Literature Review

This section covers prior work aimed at modelling text data types. The section firstly discusses differences between stand-off and in-line text data types. It then reviews existing text modelling and processing platforms, such as UIMA [20] and GATE [9], that support the design and development of text data type systems. Lastly it explores previous approaches for modelling text data type systems.

2.1 Types of Text Annotations

Annotations (i.e. text data types) can either be in-line or stand-off [32]. Figure 2 illustrates the same input document which is annotated using both in-line and stand-off annotations.

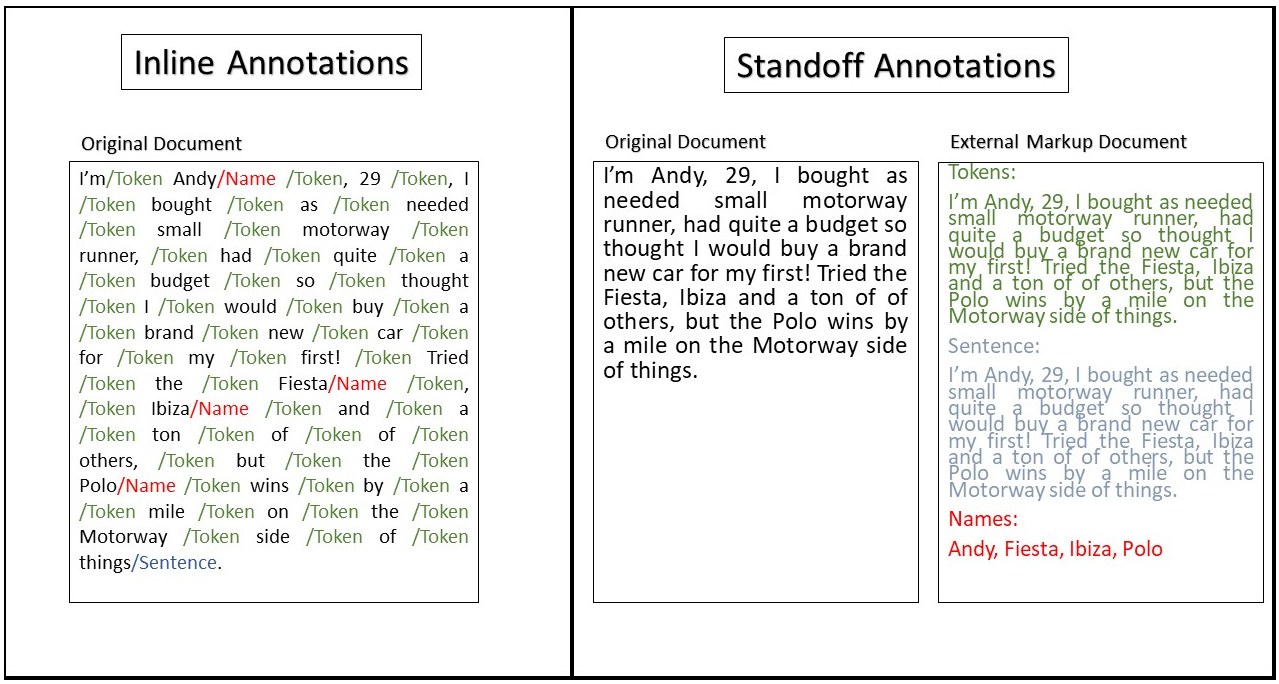

Figure 2: In-line (left-hand side panel) and stand-off (right-hand side panel) used to annotate the same input document

In-line annotations, expressed in a mark-up language such as XML or JSON, constitute a simple, convenient and straightforward way of annotating plain text documents [86]. They store the generated annotations within the same plain text document where the natural language text (i.e. input to text processing components) is actually located. However, when an input document already contains mark-up language and it is not stored in a plain text format then the generated in-line annotations may change the format of the existing mark-up tags. A second limitation of in-line annotations is that they overload the original document with additional content which can be problematic when multiple text mining components wish to process the same document [84]. Moreover, synchronising multiple text mining components that interact with the same input document can be a challenging task when using in-line annotations [31].

In contrast to in-line annotations, stand-off annotations store the generated annotations in a separate location while annotations are linked to the input document using reference pointers. Stand-off annotations do not alter the content of the input document and they can thus be used to annotate any type of documents (e.g. plain text, XML, JSON). In addition to this, Ide et al. [31] showed that multiple text mining components can seamlessly interact with the same input document considering that each component stores the generated annotations in an isolated location. However, a potential problem of stand-off annotations is that reference pointers need to be re-computed when the content of the original document is changed [84].

Rebholz-Schuhmann et al. [70] presented a framework that defines an annotation and data exchange format for modelling different text data types, such as sentences, tokens, terms, chunks and named entities. They further defined a basic set of data types to enable interoperability between different annotation tags. They used in-line annotations and justified their use for several reasons:

1. their research was in the biomedical field where 50% of the developers uses in-line annotation to store text mining analysis results.
2. in-line annotations support different text encodings (e.g. ASCII, UTF-8, Unicode)
3. software tools that handle and visualise in-line annotations are widely available

Day et al. [17] created a new annotation tool Callisto which also supports in-line annotations. They claimed that stand-off annotations caused versioning control problems when compared to in-line annotations. In a similar vein, Ion [33] annotated a large Romanian corpus using in-line annotations to ensure compliance with their web-services and workflows.

Several studies have also explored the use of stand-off annotations for modelling text data types [5, 57, 79, 50]. Basile et al. [5] developed a large annotated corpus using stand-off annotations. Nazarenko et al. [57] introduced the ALVIS annotation format whose aim was to index stand-off annotations for developing a topic-specific search engine.

Catalyst, a framework introduced by Mardis et al. [50], provided a data model which is based on stand-off annotations. The authors highlighted that stand-off annotations are more efficient and easier to test and debug when compared to in-line annotations. Additionally, stand-off annotations simplify the development process of text mining pipelines considering that any new components added to the system would not affect existing components.

Elsewhere in the literature, Nicolas et al. [59] used a combination of both in-line and stand-off annotations to model different text data types. Document sectioning data types (e.g. paragraphs) were modelled using in-line annotations. However, the authors noted that more complex data types that annotate discontinued spans of text (e.g. co-referential expressions) are difficult to model using in-line annotations and they thus relied upon a stand-off format to store and manage these data types.

In this project we choose stand-off over in-line annotations to model text data types produced by the Typhon text processing pipelines for the following reasons. Firstly, stand-off annotations are easier to distribute within a computer cluster considering that these are detached from the raw text documents. Thus, components of a computer cluster need to exchange only the produced annotations and not the actual content of the text documents. Secondly stand-off annotations can be used to annotate any type of documents (e.g. plain text, XML, JSON or textual attributes stored in a Typhon polystore) while prior work [59, 31] showed that stand-off annotations can model a wide range of different text data types. Thirdly, the vast majority of existing text modelling and processing frameworks, which are discussed in the next section, support stand-off annotations. Lastly, stand-off annotations are data type agnostic since reference pointers can be linked to text, audio, image or

video collections. As an example, stand-off annotations for audio files may use time offsets while annotations over text may use character pointers.

2.2 Text Modelling and Processing Frameworks

Currently, there exist a number of text modelling and processing frameworks, including TIPSTER [24], Ellogon [66], GATE [10], Heart of Gold [79] and UIMA [20], to name a few. The purpose of a text modelling and processing framework is to facilitate the design and implementation of text data type systems, to aid in the development of text mining components, to foster interoperability of components by defining standard communication mechanisms and to support the execution and deployment of text mining pipelines (i.e. a chained sequence of text mining components). Table 2.2 summarises advantages and limitations of existing text modelling and processing frameworks.

2.2.1 TIPSTER

TIPSTER was designed to support the development of multilingual text mining applications and to enable communication of heterogeneous components [4]. TIPSTER offers programming APIs in different languages including C, Tel and Common Lisp. The architecture adopted by TIPSTER is document centric, where documents are stored in collections. Text data types are stored in a stand-off format and defined in a type declaration. Each text data type may consist of a number of attributes while each attribute can either be a primitive (e.g. Integer, String, Boolean etc.) or a reference to another text data type or a reference to a document. TIPSTER text data types are indexed and managed for easy access and usage for different use cases and applications. The TIPSTER framework comes packaged with pre-defined text data types and attributes to facilitate interoperability of components. Moreover, TIPSTER implements a sophisticated typed annotation model which was adopted by other platforms such as UIMA, GATE and Ellogon [4]. However, TIPSTER offers limited workflow managements functionalities which in practice means that coordination of components into pipelined applications becomes challenging. Moreover, standardised parallelisation, distribution of processing and language resources are not supported by TIPSTER.

TIPSTER has been previously used by prior work for modelling text data type systems. As an example Liddy et al. [46] presented a TIPSTER-based type system consisting of paragraph, sentence and discourse-level (e.g. current event, past event, opinion, potential future event) types.

2.2.2 Ellogon

Another multilingual and general purpose text modelling and processing framework is Ellogon [4]. It supports a wide range of languages using the Unicode encoding format [4]. Ellogon is built around three basic subsystems:

- The Collection and Document Manager (CDM), which collects documents consisting of textual data and text data types are stored in a stand-off format. Annotations and attributes contain identifiers which are user-defined.
- Ellogon establishes relations between different components to automatically create pipelined applications.
- A component system which loads and integrates processing resources (i.e. modules) at runtime. Modules contain the implementation part of text mining components and declare the metadata for the framework.

Table 1: Advantages and Disadvantages of the Platforms

Framework	Advantages	Disadvantage
TIPSTER	Sophisticated annotation system	(1) No parameter or resource management and no great workflow management (2) No interchangeability, standardized parallelisation, distribution of processing and language resources
Ellogon	it offers the possibility to use components as web services	(1) it only serial and cascaded workflows are supported (2) it does not have any analysis awareness or resource management and creating parameters is very basic (3) it cannot inherit existing type definitions for annotations or reuse metadata
GATE	GATE has many GUI tools, data access structures, language resources and import filters for common document formats.	(1) Lack of a sophisticated workflow management (2) No formal specification for individual resource management (3) Type inheritance is not possible
HoG	(1) Workflows are implemented in a sequential, parallel and iterative execution (2) It does not impose a specific DTD or Schemata for annotations	(1) No parameter or resource management (2) Conditional workflows is not supported (3) Expensive transformation algorithms for XML formats
UIMA	(1) It is the most evolved and comprehensive architecture available up to now (2) It is Apache licensed and is standardized (3) It provides parallelisation and distribution of processing resources (4) It is also the only architecture that provides resource management and analysis aware parameter handling	(1) Analysis awareness of descriptor metadata (2) Lack of definition of data resources and access structures for processing resources, document and annotation model type systems (3) Nested or cascaded workflows are not allowed on the descriptor level. (4) Information cannot be accessed by other components if not specified by the architecture itself.

One disadvantage of Ellogon is that only serial and cascaded workflows are supported, while parallel, conditional, nested or iterative workflows are not. Additionally, it does not have any analysis awareness or resource management and creating parameters is very basic. Moreover, it cannot inherit existing type definitions for annotations or reuse metadata.

Petasis [65] used Ellogon to define a text data type system of tokens and named-entities such as the person's name and company's name. Likewise, [36] presented an Ellogon-based type system consisting of morphological (i.e. part-of-speech annotation), phrase (e.g. syntactic annotation, named entities, sentiment analysis, co-reference) and semantic (e.g. OWL ontology) types.

2.2.3 GATE

The General Architecture for Text Engineering (GATE) was created to provide an infrastructure for different language engineering activities [16]. It consists of easy to use GUI editing tools. There are two central elements in GATE:

- The GATE Document Manager (GDM) follows the TIPSTER specifications. Consequently, the collection of documents which have text and annotations can be modified online. The processing resources receive the annotated documents from the GDM and then return them to other processing components. Multiple annotation graphs per document can be created from one span of text.
- A Collection of Reusable Objects for Language Engineering (CREOLE) a library of processing and language resources and data structures for general usage. CREOLE API can be used by users to extend CREOLE objects while CREOLE objects are initialised and applied to documents using the Gate Graphical Interface (GGI). The CREOLE components must specify their configuration (either in XML or in Java Annotations) which is subsequently used by GATE to construct pipelined applications.

GATE has many GUI tools, data access structures, language resources and import filters for common document formats. Conditional processing and collection level processing are supported. However, iterative, nested or parallel processing are not supported by GATE due to the lack of a sophisticated workflow management system. GATE resources are separated and described using metadata which can be composed in the workflows. Inheritance is facilitated and a document model with typed annotations is well defined. Some disadvantages of GATE include its lack of a sophisticated workflow management system while the individual resource management is not formally specified and the type inheritance is not possible.

GATE has been used for developing text data type systems by several previous studies [85, 10]. Volker et al. [85] presented an ontology tool for querying Spanish linguistic data content. Their GATE-based text data type system consisted of: tokens, sentences, POS-tags, Lemmas, and Jape transducer (for shallow parsing). Bontcheva et al. [10] investigated the use of Gate for automatically identifying customisable named entities from text such as person, date, money, organisation, percent and location type.

2.2.4 Heart of Gold (HoG)

Heart of Gold was developed to create and combine annotation produced by multiple components in multilingual environments. Text data types are stored in XML stand-off markup [79]. Heart of Gold has a special focus on facilitating the development of shallow and deep parsing workflows. Heart of Gold does not impose a specific document type definition or predefined schemata for annotations. Consequently, there are no explicit constraints on how data is stored. The core component of Heart of Hold is the Module Communication Manager which deals with the requests from components and it is responsible for returning results back

to the components. The Module Communication Manager also organises workflows of processing resources, the persistence layer and any data exchange that occur between components. Workflows are specified using the System Description Language allowing sequential, parallel and iterative execution. Heart of Gold offers no parameter or resource management and conditional workflows are not supported. Additionally, expensive transformation algorithms are required due to the use of XML formats

Buitelaar et al. [13] presented an ontology-based information extraction system which uses Heart of Gold. Their information extraction system automatically identifies named entities relevant to football while their underlying text data type system includes: persons, locations, numerals, and date\time types. It also includes some sub-entity types such as actors in football (i.e. trainer, player, referee). Additionally, football-specific events (i.e. shots, headers), match events (i.e. goal, card) and match results are modelled. Similarly, Schafer [79] used Heart of Gold to annotate documents. Their type system consists of tokens, chunks, POS-tags and syntactic types identified by a shallow parser.

2.2.5 UIMA

The Unstructured Information Management Architecture (UIMA) is a framework developed by IBM that aids in the analysis of unstructured content including text, video and audio data [4]. It allows reuse of analysis components and reduces duplication of the development process considering that UIMA applications are modular (i.e. pipelines), consisting of building blocks (i.e. text mining components) that can be re-used across different pipelines. Similarly to other existing text modelling and processing framework, text data types are stored in a stand-off format. UIMA supports a very flexible typed annotation model where text mining components provide clear metadata about expected inputs (i.e. tokens, sentences) and outputs (i.e. emotion or polarity labels). It has enterprise-ready code making the execution and deployment process of text mining pipelines easier. Moreover, UIMA has demonstrated scalability and interoperability while programming APIs are available in Java and C++. A number of existing studies has demonstrated that UIMA-enabled pipelined applications can efficiently address a number of complex text mining tasks [6, 40, 7]. As an example, Kolluru et al. [39] evaluated different pre-processing components, e.g. tokenisers, using reconfigurable UIMA pipelines to optimise the performance of a chemical named entity recogniser. Batista-Navarro et al. [6] developed sophisticated machine learning-based UIMA pipelines that are trained via active learning to identify disease names in text while Kontonatsios et al. [40] showed that UIMA pipelines can be used to automatically process content in different languages and in different modalities (e.g. text and audio). Many researchers have used UIMA for textual classification purposes, for example in Sohn and Savova [81] where they used UIMA to classify the smoking status of patients. They used text analysis components from Mayo's cTAKES UIMA repository [78] (e.g. tokeniser, sentence boundary detector, document classifier).

The UIMA architecture consists of several central elements:

1. The Common Analysis Structure (CAS) constitutes the common communication mechanism of UIMA components. Essentially CAS is a data structure used by different components to read/write analysis results and it is shared among the components of a pipeline.

2. Type System Model: A hierarchy of text data types. Objects stored in CAS are formally defined in UIMA type systems while type systems are developed in Ecore (i.e. modelling language used in the Eclipse Modelling Framework). UIMA only provides an abstract-level type system leaving the actual implementation of specific text data types to third-part developers. An *Annotation* is a predefined UIMA data type which contains reference pointers (e.g. offsets) to an input document and references to other data types.

3. Abstract Interfaces: A set of standard method signatures defining different operations which users can implement.

4. Behavioural Metadata: provides information of the input/output requirements of text mining components and describes component dependencies. It includes the input text data types that a component can process, text data types that a component generates, and any other pre-requisite components that need to be executed before the component described in the Behavioural Metadata.

5. Processing Element Metadata: describes identification, configuration and behavioural information about processing components and pipelines (analytics and flow controllers). This is related to the specification of the Behavioral Metadata.

6. WSDL Service Descriptions: UIMA allows publication of pipelines as web services. WSDL describes the abstract interfaces. The binding is defined to a concrete SOAP interface (i.e. a messaging protocol allowing programs to run on different operating systems). This must be implemented by compliant architectures.

Bank et al. [4] provided a comprehensive literature review of existing text modelling and processing framework and concluded that UIMA is the most evolved and comprehensive framework available up to now. They further highlighted that UIMA is an open-source Apache project which is compatible with different standards including XMI, Ecore, XML, OCL or BPEL. Moreover, UIMA enables parallelisation and distribution of processing resources and tt is also the only framework available that provides resource management and analysis aware parameter handling.

However, UIMA presents several limitations according to the following:

- UIMA is not analysis aware of descriptor metadata as there is no applicable model for metadata inheritance. However, metadata can be provided using Java annotations.
- there is a lack of definition of data resources and access structures for processing resources, document and annotation model type systems.
- the UIMA type system is not comprehensive enough as it implements only abstract data types.
- aggregated workflows are just defined as collections of different analytics. Nested or cascaded workflows are not allowed on the descriptor level. Therefore, it becomes impossible to use document and collection based analytics iteratively or processing pipelines cannot directly communicate.
- information cannot be accessed by other components if not specified by the architecture itself.

2.3 Type-Systems and Text-Types

There are many studies on UIMA, however, not many illustrate their UIMA text data type systems in detail. A few studies which have described their type systems are Hahn et al. [29], Wu et al. [88],Kim et al. [38] and Pon et al. [68].

Hahn et al. [29] presented an UIMA-based type system covering general-purpose text data types, including morphological, syntactic and semantic types. The authors demonstrated that their UIMA-based type system could annotate the entire cycle of text mining analysis with most of core types. They further provided an extension to the bio-medical domain by modelling domain-specific text data types.

Likewise, Kim et al. [38] presented a text mining system, developed in UIMA, that classified the assertion type of medical problems in clinical notes. Their UIMA-based type system included linguistic attributes, lexical, syntactic, lexico-syntactic, and contextual types. Pon et al. [68] explored automatic identification of *interesting*

news articles using the UIMA framework. Their designed data type system included topic relevancy features, uniqueness measurements, source reputation, freshness, subjectivity, and polarity of news articles.

Wu et al. [88] defined a UIMA type system for clinical text mining that enables interoperability between structured and unstructured data generated in different clinical settings. The authors modelled several structured data-types (i.e. documentid, metadata, demographic, sourcedata), utility types (i.e. pairs, pair, probability-distribution), and text span types (i.e. paragraph, sentence, list, segment, document, segment). They also defined syntactic types including different types of tokens (i.e. NumTokens, Newline-Tokens, Punctuation-Tokens,Base-Tokens), Lemmas, POS-tags, and Treebank-Nodes. Moreover, they covered semantic types and subtypes including a persons title, times and relations between annotations. They included deep semantic types and a model of core CEMs and additionally the relation namespace, with both text relations (spanned) and referential semantic (unspanned) relations.

Table 2 summarises the most common text data types developed by prior work using the UIMA framework. We have further classified existing data types into the following three categories: a) General types denoting the textual unit that text mining components annotate (i.e. Document level, Paragraph level, Sentence level, Phrase level, and Token level), b) Syntactic types (e.g. pos, chunk, syntactic dependency tags) that annotate syntactic properties of documents and c) Semantic types (e.g. polarity, named entities, events) which aim to model information relevant to the meaning of documents.

In this project, we design and develop the Typhon text data type system in UIMA. We choose UIMA over other existing text modelling and processing frameworks for the following reasons. Firstly, UIMA type systems support data type inheritance. In practice this means that an existing UIMA type system can be readily extended with new data types. Moreover, data type inheritance accelerates the development process of type systems considering that children data types need to implement only those attributes that are not present in their parent data types. Secondly, previous work showed that UIMA-based type systems can be used to annotate different types of data (e.g. text, audio) [40]. Thus, our Typhon data type system can be extended in the future to other data modalities apart from text. Thirdly, UIMA offers well-defined communication mechanisms (i.e. CAS) to support interoperability of text mining components. This becomes important to the Typhon project since standard pre-processing components of our text processing pipelines will use existing text mining libraries and tools. Finally, UIMA implements a cluster controller[1] to enable large-scale processing of big data collections which is one of the core research objectives of our work.

[1] uima.apache.org/doc-uimaducc-whatitam.html

Table 2: Existing Data Types

	Types	Examples/Details	Authors
General	Document	language	[88]
	Paragraph	A few sentences with a space after	[88]
	Sentence	Three or more words usually ending with a full-stop or punctuation	[88, 28]
	Token	Words, numbers, punctuation	[88, 28]
	Phrase	"Throw in the towel" means to give up	[77]
Syntactic	N-gram	One or more word split	[60]
	POS-tags	Verb, nouns, adjectives	[88]
	Lemma	Studies-Study	[88]
	Stem	Studies-Studi	[28]
	Syntactic dependencies	Alice Saw Bob=>(NOUN-VERB-NOUN)	[28]
Semantic	Named Entities	Persons name title	[88, 28]
	Polarities	Positive,negative, neutral	[72]
	Facts	"The president is Trump"	[81]
	Events	“Hit the car due to the faulty brakes”	[81]
	Domain	Product-ID, Weather, Cars	[88]
	Negations	No, not, neither, never, no one, nobody, none, nor, nothing, nowhere	[81, 77]

3 Typhon Text Data Type System

In this section, we describe the Typhon text data type system. The proposed type system was developed by taking into consideration the different requirements of the Typhon use case partners found in deliverable D1.1 (project requirements are further discussed in section 5). The text data type system described here consists of general purpose text data types that can be used by all use case partners across a wide range of different applications and can be easily extended or edited in the future to accommodate new requirements that may arise. Some use case partners, such as ALPHA Bank, had specific requirements for modelling information extracted from text based on specialised data types that are only applicable to their underlying domain. We have thus extended our general purpose Typhon type system by creating domain-specific text data types which are discussed in section 5.

3.1 Overview of developed Type systems

Tables 3 to 6 summarise various information of the developed Typhon text data types, including a) the name of the data type, b) whether the data type is applicable to the use case requirements provided by the four use case partners of Typhon (i.e. VW, ALPHA, NEO, GMV), c) the text unit of annotation and d) the attributes of the data type. As mentioned earlier in Section 2, the Typhon text data types are organised into three categories, namely general, syntactic and semantic. General data types simply partition an input document into segments (i.e, document, paragraph, sentence, phrase or word) which form the basis for further analysis by syntactic or semantic components. Syntactic types model the syntactic structure of documents (e.g. identifying pos tags). Syntactic types annotate words or phrases with syntactic information. Semantic types model different semantic properties of an input document such as the document category (e.g. sports, news, politics etc), the polarity of the document (e.g. positive/negative/neutral), the named entities occurring in the document (e.g. named of organisations, locations) etc. Semantic types annotate different textual units, e.g. named entities annotate words or phrases, events and relations annotate sentences, topics and categories are document-level data types while the co-reference type is a multi-level annotation considering that co-referring pairs (e.g. Barack Obama/He) may span across the same sentence or paragraph or across multiple sentences or paragraphs.

The Typhon data types are organised into a hierarchical system which is illustrated in Figure 3. *Top* and *Annotation* are abstract-level data types defined by the UIMA framework. *Top* is an empty data type and it is simply used to denote the top level of the hierarchy while the *Annotation* data type defines reference pointers to the input document (i.e. begin and end offsets). All Typhon data types extend the *Annotation* type in order to inherit the reference pointer attributes. It should further be noted that a data type can be used as an attribute in a second data type. As an example, the *Dependency Parsing*, which links POS tags into syntactic relationships, contains as an attribute a list of constituent POS data types.

Table 18 in AppendixA illustrates different text processing pipelines that are used to generate the Typhon text data types. Moreover, the table shows the structure of the text data types stored in an ElasticSearch back-end.

3.1.1 General Data Types

In the following list, we describe our justification for modelling different text data types. These include requirements from the use case partners and how they can support structuring and managing textual data.

1. ***Document***

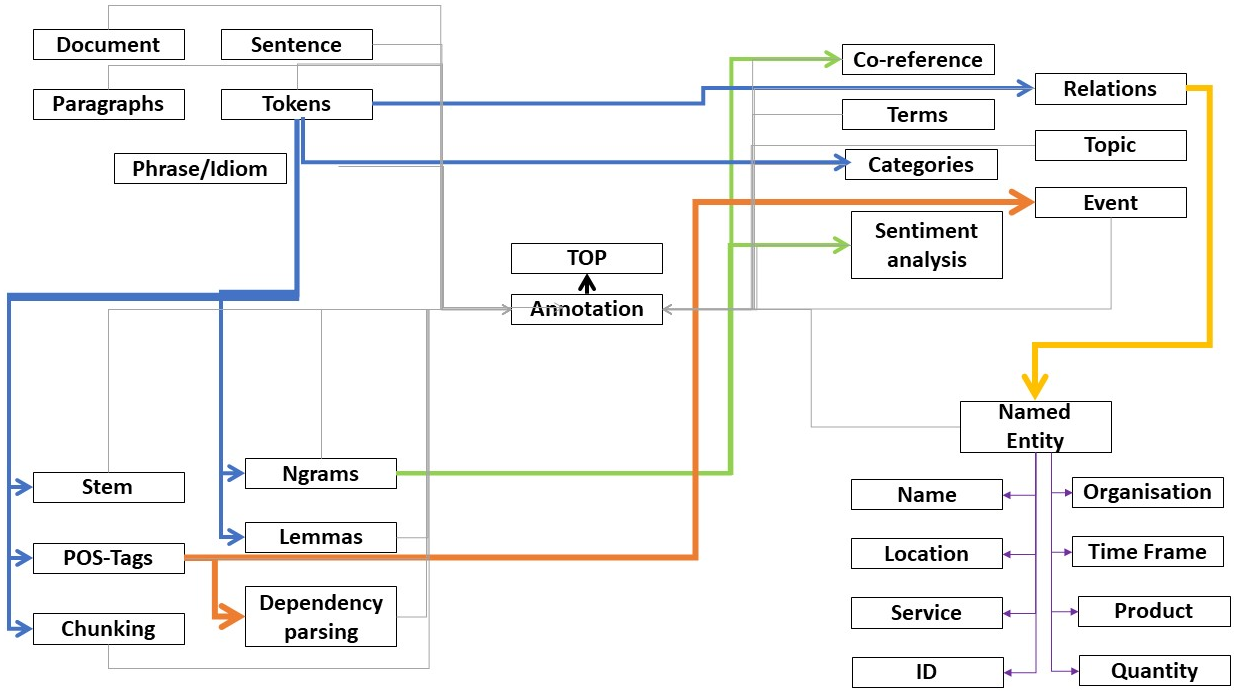

Figure 3: Type System

Table 3: Typhon General-purpose Text Data Types

Data Type Name	VW	ALPHA	NEO	GMV	Text unit	Attributes
Document	x	x	x	x	Document	+ begin: Int + end: Int + language: String + encoding: String + sourceFile: URI
Paragraph	x	x	x	x	Paragraph	+ begin: Int + end: Int
Sentence	x	x	x	x	Sentence	+ begin: Int + end: Int + sentenceValue: String
Token	x	x	x	x	Word	+ begin: Int + end: Int + tokenValue: String
Phrase	x	x	x	x	Phrase	+ begin: Int + end: Int + phraseValue: String

Table 4: Typhon Syntactic Text Data Types

Data Type Name	**VW**	**ALPHA**	**NEO**	**GMV**	**Text unit**	**Attributes**
N-gram	x	x	x	x	Word /Phrase	+ begin: Int + end: Int + bigramValue: List<2> (Token, Lemma or Stem) + trigramValue: List<3> (Token, Lemma or Stem)
POS-Tags	x	x	x	x	Word	+ begin: Int + end: Int + posValue: String
Lemma	x	x	x	x	Word	+ begin: Int + end: Int + lemmaValue: String
Dependency parsing	x	x	x	x	Word	+ begin: Int + end: Int + sourceNE: NamedEntity + targetNE: NamedEntity + label: String
Chunking	x	x	x	x	Word	+ begin: Int + end: Int + constituentTokens: List<Token> + constituentPOS: List<POS> + label: String
Stem	x	x	x	x	Word	+ begin: Int + end: Int + stemValue: String

Table 5: Typhon Semantic Text Data Types

Data Type Name	**VW**	**ALPHA**	**NEO**	**GMV**	**Text unit**	**Attributes**
Sentiment Analysis	x	x	x	x	Document /Sentence	+ begin: Int + end: Int + polarityLabel: String + polarityScore: Float
Co-reference	x	x	x	x	Multi-level annotation	+ begin: Int + end: Int + antecedent: List<Token> + anaphor: List<Token>
Relations	x	x	x	x	Sentence	+ begin: Int + end: Int + sourceNE: NamedEntity + targetNE: NamedEntity + label: String
Categories	x	x	x	x	Word /Document	+ begin: Int + end: Int + categoryLabel: String + categoryScore: Float
Topics	x	x	x	x	Document	+ begin: Int + end: Int + descriptiveWords: List<str> + wordCoefficients: List<Float>
Terms	x	x	x	x	Word /Phrase	+ begin: Int + end: Int + constituentTokens: List<Token> + weight: List<Float>
Event	x		x		Sentence	+ begin: Int + end: Int + triger: token + arguments: List<NamedEntity> + theme: String

Table 6: Typhon Named Entities Text Data Types

Data Type Name	**VW**	**ALPHA**	**NEO**	**GMV**	**Text unit**	**Attributes**
Name	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + nameValue: String
Location	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + locationValue: String
Product	x				Word/Phrase	+ begin: Int + end: Int + productValue: String
Time period	x		x		Word/Phrase	+ begin: Int + end: Int + timeValue: String + date: String + day: String + month: String + year: String
Organisation	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + organisationValue: String
Service	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + serviceValue: String
ID Entity	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + IDEntityValue: String
Quantity	x	x	x	x	Word/Phrase	+ begin: Int + end: Int + quantityValue : Int

The document data type consists of the following attributes: language (string) which stores the language of the document, encoding (string) which denotes the encoding format of the document (e.g. UTF-8), sourceFile (URI) which is the location, e.g. file path, of the document) where the document is actually stored, begin (integer) and end (integer) which are reference pointers inherited from the *Annotation* data type.

Language is an essential attribute for this project considering that use case partners require analysing textual data in different languages (e.g. German, Greek, Spanish, English). The language attribute is also important since the vast majority of text mining components, such as POS-taggers, sentiment analysers, dependency parsers etc., are language-dependent. In practice this means that a language-dependent component needs to identify the language of the input document (by examining the value of the language attribute) in order to load the correct configurations for the underlying language of the document. The encoding is also important because data can be stored in different encoding formats and thus the text mining processing components need to know the correct encoding format of an input document. Specifying the encoding format allows other components to function properly (e.g. identifying the text characters correctly). The last feature is the data size. Data-size is fairly important and can be used for data verification purposes and for partitioning a large file into smaller partitions.

2. ***Paragraph***
   Paragraph data types have been previously used in various text mining applications. As an example, Jaruskulchai et al., [34] presented an automatic summarisation system that used the first sentence of each paragraph to summarise an input document. Paragraph data types have also been used for topic detection and opinion summarisation [43].

   The paragraph data-type has the following attributes: begin (integer - beginning of the paragraph), end (integer - end of the paragraph), and paragraphValue (string - actual textual content of the paragraph).

3. ***Sentence***
   A sentence data type marks the character offsets of sentences in a given documents. It is an essential data type for text classification purposes including sentiment analysis. Similar to the paragraphs, it has also been found useful in topic detection [43]. There are various way to extract sentences. Punctuation (i.e. question mark or exclamation mark) could be used for sentence segmentation, however this could be challenging in some languages [75]. In Spanish for example, you could use punctuation to confirm a statement such as ¿No? this is equivalent to “am I right? ". The end of the sentence will be at the second question mark not the first. However in English, usually question mark or exclamation marks indicate the end of a sentence. Additionally, due to encoding issues, there can be hidden characters such as ‘\n’ and ‘Â’ which split the sentence, these may cause issues if not dealt with in a correct manner. Lastly, some users may not use punctuation correctly and may not end sentences with a full-stop or punctuation, this is most common in reviews and informal text [80]. To address these issues the sentence type should take into consideration the source of the data and the language.

   The sentence data-type similar to the paragraph data type consists of the attributes: begin (integer - beginning of the sentence), end (integer - end of the sentence), and sentenceValue (string - textual content of the sentence).

4. ***Token***
   The token is the most common data type it is used by the vast majority of text mining components (e.g. stemmer, n-grams, term extraction, POS taggers etc.) [15, 77, 7]. Token data types denote the boundaries of words within a document. There are different ways to tokenise a document: splitting at whitespace or splitting a document into its constituent words. In this project, we experiment with

2 different tokenisers: Standard tokeniser which is our own Java-based implementation which simply splits words according to white spaces and the StringTokenizer class from the java util package[2]. A full report of the results are included in section 6. In summary, we found that our tokeniser led to more accurate results when compared to the StringTokenizer.

Tokens can be weighted by different weighting schemes, such as the term frequency (TF) and the term frequency–inverse document frequency (TF-IDF). TF and TF-IDF assign a weight to each token which denotes the importance of that token in a document. TF and TF-IDF have been interchangeably used in document classification approaches. Haddi et al. [26] explored the differences between Term Frequency, TF-IDF, and Feature Presence (FP) (i.e. number of tokens present in a document). They found that document classification models using TF-IDF led to a superior performance.

Tokens are important for this project as GMV needs fast text indexing and keyword search. Token data types usually form the basis of larger data types such as keywords. Moreover, all four use case partners require named entities which consist of a sequence of tokens, e.g. multi-word phrases. A token data type contains similar attributes to both the paragraph and sentence: begin (integer - beginning of the token), end (integer - end of the token), and tokenValue (string - textual content of token).

5. ***Phrase/idioms***
   Many languages have phrases and idioms which refer to a certain meaning. For example, the phrase “It is raining **cats and dogs**" refers to heavy rain. If this phrase is not taken as a whole it could change the meaning that the sentence is trying to convey. Obviously, phrases in one language could not have the same meaning in another. Therefore, language should be identified before detecting phrase data types. Usually phrase and idiom extraction are applied to informal textual data such as feedback. Phrases can be used in sentiment analysis or topic detection [87]. Phrases can be extracted from textual data using a manually constructed lexicon.

   Each phrase or idiom consists of the attributes: begin/end reference pointer, and phraseValue(string - textual content of that phrase\idiom).

3.1.2 Syntactic Data Types

Syntactic data types are important for modelling the syntactic structure of an input document. Moreover, syntactic data types are used as constituent attributes in larger semantic text data types. The following list present the syntactic data types of our type system.

1. ***N-gram***
   N-grams are useful data types and can be used for many purposes. They can be used as input to machine learning classifiers that automatically detect different semantic types (e.g. polarity, category of a document). N-grams can also be used for identifying multi-word expressions such as “New York". Some existing n-gram extraction tools, such as the in-built text processing functionalities of the WEKA platform, consider every possible combination of tokens as valid n-grams. An example consider the following sentence “I like this book", if we required bigrams (i.e. sequence of two tokens), the bigrams extracted by the Weka library would be: “I like", “like this", “this book", “I this", “I book", “like book". In another more simplified version, it could be just: “I like" and “this book". N-grams are required for some named entities such as detecting the full person’s name: first name and surname.

   N-grams can be directly computed from tokens, saving computational time since the document collection does not have to be loaded into memory again. They can also be derived from the base forms of

[2]https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html

tokens such as lemmas or stems. The N-gram data type consists of the following attributes: begin/end reference pointer, and bigramValue and trigramValue (List - constituent tokens of the n-gram).

2. ***Part-Of-Speech (POS)-Tags***
   A part-of-Speech tagger assigns part-of-speech tags (i.e. noun, verb, adjective) to each word. POS-tags provide useful information to semantic analysis components such as text classification or sentiment analysis and they are used as constituent attributes in larger data types (e.g. dependency parsing type). A standard set of POS tags consists of the following: adjective (ADJ), adposition(ADP), adverb (ADV), conjunction (CONJ), determiner (DET) , noun (NOUN) , numeral (NUM) , particle (PRT), pronoun (PRON) , verb (VERB), punctuation marks (.), and other (X). There are many libraries that provide POS-tagging functionalities (e.g. LingPipe, Stanford, LBJ, FastTag, and OpenNLP) [35, 48, 92]. Some libraries include more refined categories of POS tags such as the Penn treebank library. The verb in this library has three forms: base form (VB), past tense (VBD) or present participle (VBG). In this project, we will implement POS tagging pipelines using the Standford CorNLP library due to its capability to process different languages (i.e. Arabic, Chinese, German, French, Spanish) which is required in this project [48].

   A POS data type contains the following attributes: begin/end reference pointer, posValue (string the POS tag).

3. ***Lemma and Stem***
   Lemmatisation and Stemming aim at identifying normalised forms (root) of words. The difference between lemmatisation and stemming is that a stem (output of stemming) might not be an actual word but a lemma (output of lemmatisation) is an language word. As an example, consider the word ‘Studies’, its stem form is ‘Studi’ and its lemma is ‘Study’. Stemming is the most widely applied morphological technique for information retrieval [41]. Stemming benefits include reducing the total number of distinct index entries and query expansion by bringing word variants and derivations together [41].

   Stemming is known to be challenging for languages that present high variability of verbs (such as Spanish, Italian, Catalan, Slovenians) while the poor performance of stemming component may lead to loss of information and the resulting annotations may not be refined enough [83]. A second limitation of stemming is that words that are semantically different may be incorrectly grouped under the same stem cluster, for instance, in Spanish estafa (swindle) and estafeta (post office) are both grouped under the stem cluster ‘estaf’. On the other hand, lemmatisation correctly groups these two semantically different words under two different lemma clusters: estafa and estafeta. Another issue with stemming is that it could give different stems for the same verb. As an example, the word resolver (to resolve) and resuelto (resolved) could be ‘resol’ and ‘resuel’. The only valid lemma here should be resolver. Lemmatisation was also found more beneficial (i.e. better performance) in a comparison study that evaluated lemmatisation against stemming for the Finish language [41]. Other studies reported that lemmatisation requires substantial computational resources when compared to stemming and they thus preferred stemming over lemmatisation to derive the root form of words [14, 3]. Also, stemming is a dictionary-independent method which can deal with unknown words. On the other hand, lemmatisation requires a large dictionary which usually is not available for any language or domain.

   Stemming and lemmatisation can be beneficial for this project due to the partners’ requirement for fast keyword search like GMV, spell checking required by VW, and sentiment analysis required by both GMV and VW. Stemming allows us to reduce running time (i.e. faster searches by reducing the variation of word matches using the root form) and makes classification or topic detection more accurate. In this project, we use both stemming and lemmatisation. Lemmatisation is generally preferred over stemming as it yields more accurate results. However, stemming is faster than lemmatisation while

stemming tools are available for a larger number of languages considering that stemming is a dictionary-independent method. We have experimented with the following stemming tools: EnglishStemmer[3] and PorterStemmer from the tartarus snowball package[4]. We found that the EnglishStemmer led to a better performance in terms of accuracy and running-time. More details of these results will be presented in section 6.

A stem data type consists of the following attributes: begin/end reference pointer and stemValue (string - stem form of word). Likewise, the lemma data type consists of: begin/end reference pointer and lemmaValue (string - stem form of word).

4. ***Dependency parsing***
A dependency parser analyses the grammatical and syntactical structure of a sentence and it identifies syntactic relationships between words. Dependency parsing focuses on relationships between words while the phrase structure focuses on identifying phrases and their recursive structure. Mcdonald et al. [51] presented a collection of treebanks with syntactic dependencies for several languages: German, English, Swedish, Spanish, French and Korean. They made this universal treebank freely available. Dependency parsing is a challenging tasks due to language ambiguities. For example, consider the following sentence "Visiting relatives can cause problems". Here, a dependency parser identifies two possible syntactic structures where "visiting" can either be an adjective or a verb in the sentence.

Identifying relationships between words may facilitate more meaningful keyword-based search as multi-word queries will only retrieve documents where the query words have a syntactic dependency. For our project, we chose to use the Stanford Parser due to its capability to handle multiple languages (i.e. Arabic, Chinese, German, French, Spanish).

A dependency parsing data type consists of the attributes: begin/end reference pointer, label (string - label of syntactic relationship between two tokens), sourceNE (Token - the source token of the syntactic relationship) and targetNE (Token - the target token of the syntactic relationship).

5. ***Chunking***

Chunking also known as shallow parsing produces data types which are similar to n-grams. However, chunking is implemented on top of POS tags. Chunking takes as input POS tags and clusters the pos tags into larger syntactic units (e.g. noun/verb phrases). There are standardised sets of tags which are similar to POS tags, for instance: Noun Phrase(NP) and Verb Phrase (VP). There are certain syntactic rules for chunking. For example, the Noun Phrase (NP) chunk can be formed when the chunker finds an optional determiner followed by any number of adjectives and then a noun. Chunking is implemented by different publicly available tools such as Spacy (python) or TextBlob [71]. NLTK also provides regular expressions for generating chunks [71]. Chunking has certain advantages over dependency parsing according to Osenova et al., [61]. Dependency parsing is not very robust or cheap to compute while chunking is faster and may produce more reliable results.

Chunking is useful for our project as it can be used as a pre-processing component in named entity recognition pipelines. Osenova et al., [61] showed that chunking improves upon the performance of a named entity recogniser considering that the resulting syntactic units produced by a chunker can be used by a named entity recogniser for further processing. In this project, we will use the OpenNLP library [5], for developing chunking components. A chunk data type consists of the following attributes: begin/end

[3]http://snowball.tartarus.org/algorithms/english/stemmer.html
[4]http://snowball.tartarus.org/algorithms/porter/stemmer.html
[5]http://opennlp.apache.org

reference pointer, constituentTokens (List<Token> - tokens that form a chunk unit), constituentPOS (List<POS> - pos tags of the constituent tokens of a chunk unit), label (string - the label of the chunk unit).

3.1.3 Semantic Data Types

Semantic data types are related to the meaning of text. Extracting semantic data types can be realised using either rule-based approaches (e.g. regular expressions) or machine learning-based methods. In the list below, we outline the semantic data types we created for this project.

1. ***Sentiment analysis***
   Sentiment analysis is used to analyse opinions expressed in textual content. It is popular nowadays due to the vast amount of on-line textual sources (e.g. tweets, blog/facebook posts, product reviews). Companies are using sentiment analysis to inform business decisions [47] while political parties often use sentiment analysis to better understand citizens' opinions and to subsequently aid more informed political campaigns [55]. Sentiment analysis has been previously used across a wide range of different textual sources relevant to movies, products, tourism, education [2], sports [90], and politics [55]. Some of these domains require sentiment analysis results in real-time. For instance, sentiment analysis can be used by a University lecturer to obtain instant student feedback [2]. Organisations and businesses could also be interested in analysing customers opinions to address issues in a timely manner [63].

   A sentiment analysis pipeline requires a number of pre-processing components, including tokenisation, weighting tokens using either the TF or the TF-IDF scheme, stemming, n-gram extraction and POS-tagging. In this project, we develop and evaluate sentiment analysis pipelines in UIMA which are discussed in detail in in section 6.

   Sentiment analysis is important for our use case partners: VW, GMV and Alpha Bank have customer reviews and customer feedback collections and they wish to better understand opinions expressed in these collections.

   A sentiment analysis data type has the following attributes: begin/end reference pointer, polarityLabel (string - the polarity tag), and polarityScore (Float - the polarity score).

2. ***Co-reference resolution***
   Co-reference resolution aims to identify two or more words or phrases that refer to the same entity (e.g. person/organisation/location). An example for this is "Sarah said she will have it", 'Sarah' and 'she' refer to the same person. Co-reference resolution has been used for question answering [54], automatic summarisation [22] and named entity extraction [11]. There are many tools that implement co-reference resolution such as Stanford CoreNLP and LingPipe [58]. Standford CoreNLP is more common and widely used among the text mining community [45]. It has also been adapted for different languages. Three different co-reference resolution systems are available in Stanford CoreNLP: a fast rule-based deterministic method, a machine learning-based and a neural network-based model. The neural network-based method have been previously shown to achieve a superior performance [25].

   Co-reference resolution is important as it can improve upon the accuracy of named entity identification by finding the co-referring words/entities. For this project Stanford CoreNLP will be used due to its superior performance reported in the literature[49].

   A Co-reference resolution data type has the following attributes: begin/end reference pointer, antecedent (List<Token> - the main entity referred in text), anaphora (List<Token> - a word or a multi-word phrase that refers to the antecedent).

3. ***Relation***
   Detecting relations between named entities is an important task in text processing. These relations could link instances of the same named entity or instances of different named entities. Examples of relations are: *Organisation* **located in** *Place*, *Person* **employed by** *Organisation*, *Organisation* **part of** another *Organisation*, and *Person* **married to** *Person*. Automatic identification of relations extracts meaningful information from text which can subsequently be used to aid more efficient search. There are several methods for extracting semantic relations: hand-built patterns, bootstrapping methods, supervised methods, distant supervision, and unsupervised methods [37]. For our project, we will use the Stanford Relation Extractor which is part of the Stanford coreNLP library.

   The relation data type consists of: begin/end reference pointer, sourceNE (NamedEntity - the source named entity participating in the relation), targetNE (NamedEntity - the target named entity participating in the relation), and label (string - the name of the relation).

4. ***Category***
   Automatic document categorisation is a general case of semantic analysis that aims to assign a relevant category (from a predefined list of categories) to an input document. An example of document categorisation is: *I wonder who will win the world cup*. This would be classified with the category 'Sport'.

   Automatic document categorisation will be used in this project to group documents according to their common category. After assigning categories to documents, a user of the Typhon platform will be able to form faceted queries in order to retrieve documents relevant to a pre-specified category. We will implement automatic document categorisation pipelines using a combination of machine learning and rule-based methods to improve upon the performance of our pipelines.

   A category data type has: begin/end reference pointer, categoryLabel(string - name of the category) and categoryScore (double - a confidence value that determines how likely it is that a document belongs to a specific category).

5. ***Topics (by unsupervised topic modelling)***
   Topic modelling is an unsupervised method, i.e. it does not require hand-annotated examples for training the machine learning model, that aims to identify topics within a collection of documents. A topic is described by a set of descriptive keywords that can be used as indexing terms for efficient topic-based search. In topic modelling a cluster of descriptive keywords is created for each topic while multiple topics are assigned to every document. Topic modelling has been previously shown to improve the accuracy of automatic document categorisation by grouping similar words together in topics instead of using each word as a feature [67]. Moreover, topic modelling has been used for developing topic-based search engines where users search a database using topic-based queries instead of keyword-based queries [69].

   Topic modelling algorithms include: 1) Latent Dirichlet Allocation (LDA) [8] 2) Latent Semantic Analysis, and 3) Non-Negative Matrix Factorization (NMF) [18]. In this project we will use the MALLET [21] library which offers an efficient implementation of the LDA algorithm.

   A topic includes the following attributes: begin/end reference pointer, descriptiveKeywords (List<String> - a list of descriptive keywords for that topic) and wordCoefficients (List<Float> - a coefficient score assigned to each descriptive keyword denoting the importance of that keyword to the underlying topic).

6. ***Terms***
   Term extraction is a subtask of information extraction that automatically extracts terms from a collection

of documents. Term extraction becomes important in domains such as medicine or law where documents consist of a substantially large number of domain-specific terms.

There are three approaches used in term extraction: linguistic, statistical, or hybrid [64]. The linguistic approach identifies word combinations matching certain morphological or syntactical patterns. Consequently, dependency parsers, POS taggers and morphological analysers are used as pre-processing components in a term extraction pipeline. Terms are then identified using different pattern matching techniques. A limitation to this method is that it is language-dependent considering that terms appear in different syntactic templates across different languages [64]. The statistical method searches for repeated sequences of lexical items. Unlike the linguistic approach, statistical-based methods are language independent. Finally, a hybrid term extraction method combines both a linguistic and a statistical model to more accurately identify terms.

Term extraction is important for this project as automatically extracted terms will be used as indexing-keywords in order to provide a richer set of possible queries that a user can perform. For our project, we will use a language-independent statistical-based method considering that the use case partners require analysing documents in different languages.

A term data type consists of: begin/end reference pointer, constituentTokens (List<Token> - a sequence of tokens that are the constituent parts of the term) and weight (Float - weight value which shows the importance of a term in a document).

7. ***Event***
   An event is an occurrence of a textual fact. An event is centred around a *trigger* word which denotes the presence of an event in a document. In addition to the trigger, an event consists of arguments corresponding to the entities participating in the event. Event identification is a challenging text mining task which typically involves three sub-tasks: determining the span of text identifying an event, determining the event trigger, and determining the participants of the event. There are three ways of detecting an event from text: data-driven method which relies upon machine learning, knowledge-driven which uses patterns, rules, and expert knowledge and a hybrid approach which is combination of data-driven and knowledge-driven methods [52]. The data-driven method usually requires a large amount of manually annotated data for training a machine learning model but it can be readily applied across different domains without using specialised domain knowledge or expertise. On the other hand, knowledge-based methods require a small amount of data but they rely upon domain specific pattern matching rules which are manually developed.

   In the context of this project, we will develop event extraction pipelines using the OpenNLP[6] library. Events are useful for the use case partners considering that VW requires identifying events relevant to car faults and Neo Odos needs to better understand when and why heavy traffic may occur by analysing automatically extracted events.

   An event data type consists of the following attributes: begin/end reference pointer, trigger (Token - the token which *triggers* the even), arguments (List<NamedEntity> - two or more named entities that participate in the event) and theme (string - the type of the event).

8. ***Named Entities***
   Named Entity Recognition (NER) assigns predefined semantic categories (e.g. person names, organisations, locations) to a sequence of tokens. NER has been previously used for developing efficient semantic search engines [1, 27]. Additionally, NER can improve upon the performance of a search engine by disambiguating the underlying query keywords, for example, searching for Apple the company

[6] `https://opennlp.apache.org/`

it will dismiss results of Apple the fruit. NER can be implemented using linguistic-based techniques as well as statistical models (i.e. machine learning). Linguistic-based methods use existing lexicons and local grammar (i.e. patterns to match a named entity) [91]. Statistical-based methods use machine learning approaches and generally require a large amount of manually annotated training data [91]. To address this problem, semi-supervised methods can be used which require a significantly smaller amount of training data to accurately identify named entities in text. Hybrid methods are a combination of statistical and rule-based approaches.

A named entity can be general and fit any domain (e.g. person names) or it can be doman-specific (e.g. model of a car). The different types of named entities have to be selected according to the purpose and the requirements of each application scenario. In our project, we will use the following named entities:

(a) Person name: This can be either a customer's name, employee's name, or any person's name. A person name is usually extracted using regular expressions [56]. This is due to the fact that person names usually start with capital letters. However, such regular expressions may not by applicable to any language (i.e. in Arabic person names do not require capitalisation). Some applications require identifying full name (i.e. employee name) while others only the first name of a person (i.e. customer service help).

A person named entity type includes the following attributes: begin/end reference pointer, and nameValue (string - the actual value of the person's name).

(b) Location: It is one of the most common named entity data types in addition to person and organisation names [56]. A location named entity may refer to the location of an organisation, customer's address, or location of a car collusion. The location can be expressed in natural language text or in other formats such as GPS coordinates, latitude and longitude. A location named entity can be further divided into subtypes like country, county or state, city, street or postcode. Location identification can be performed using a dictionary of existing location names or by using readily available pre-trained machine learning models such as the Stanford Named Entity Recognizer (NER)[7]. Stanford NER is based on linear chain Conditional Random Field (CRF) sequence models (i.e. machine learning) and can be adapted to fit any domain, application or language [19].

The location attributes include: begin/end reference pointer, and locationValue (string - the actual value of the location name). There could be subtype String attributes here which include country, county, city or postcode.

(c) Organisation: Refer to names of companies, institutions, government organisations. Some examples include: Apple (company), Edge Hill University (institution), or National Institute for Health and Care Excellence (government organisation). It is important to detect the organisation named when dealing with multiple domains such as in our project considering that organisation names can help organise data into separate datasets for different purposes. The organisation attributes include: begin/end reference pointer, and organisationValue (string - the actual value of the organisation name).

(d) Product: Identifying product names in textual content is important for use case partners (e.g. VW) that deal with a number of different products and wish to efficiently search a text database using the name of a product. A product name can be general or specific and it can be accompanied by an ID or sub-type of a product. As an example consider the product *car*. A car has a model and make and a specific ID. Identifying product names has been a main spotlight for sentiment analysis as

[7] `https://nlp.stanford.edu/static/software/CRF-NER.shtml`

customer feedback is essential nowadays for both customers and producers. Identifying product names is one of the steps towards understanding customers' feedback (i.e. a requirement of VW).

The product data type incudes: begin/end reference pointer, and productValue (string - the actual value of the product name).

(e) Service: Identifying service names is important for financial institutions-ALPHA bank (loan service) or a car vendors-VW (customer repair service). A service name is less common than other named entities. However, in this project service name identification becomes essential considering that all use case partners provide different type of services to their customers.

Similarly to the previously described named entities, a service data type consists of: begin/end reference pointer, and serviceValue (string - the actual value of the service name).

(f) Time period: It is a another less common data type, however not less important when compared to the above-mentioned named entity types. Time mentions can be a date: year, month, day, but it can also be the hour of a day: hour, minute, second. Time mentions can appear in either a numerical or in a character format. Time mentions can be useful for many applications such as identifying the date of purchase, date of joining a service or company. In our project, Neo Odos is interested in identifying time mentions in order to automatically detect the date and time of the day when heavy traffic occurs in motorways. Similarly VW may need to identify the time of a car collusion to report an incident. GMV may also require time identification for specifying the time of a purchase.

The time attributes include: begin/end reference pointer, timeValue (string the actual value of a time mention), date (Date - the date of a time mention), day (string - the day of a time mention), month (string - the month of a time mention), year (string - the year of a time mention).

(g) ID Entity: This data type identifies all the records that are unique. Examples include: customer or employee IDs. This is similar to the Uniqueidentifier data type in SQL. The attributes include: begin/end reference pointer, and IDEntityValue (string - the actual value of the ID entity name).

(h) Quantity: This a very generalised data type that represents any quantity value which is an integer. It can be used for many reasons such as quantity of products, the amount of money in one's bank account, or a total sum of products purchased. Its attributes include: begin/end reference pointer, and quantityValue (int - the actual value of the Quantity type).

In addition to general data types which are described in this section, there exist specialised data types which are relevant to only one domain or use case partner. Specialised data types are described in section 5.

4 Extensibility and Flexibility of Typhon Type System

UIMA type systems are flexible and readily extensible with additional data types[4]. UIMA type systems structure data types in a hierarchy governed by *is-a* relationships between types and subtypes. Therefore, new data types can extend existing data types without having to re-model attributes that already exist in inherited types. This is important for a project where new requirements have to be integrated frequently. The flexibility of the UIMA framework stems from the fact that the text processing components can be freely combined into text processing workflows using a shared type system of common data types.

The TyphonML type system of textual data needs to be extensible for several reasons. Possible extra text processing requirements from Typhon use case partners can be added easily in the future, without re-creating a new type system from scratch. Other applications or users may require new types, to cover domains other than those of the use case partners, such as health, sports or education.

According to requirement D5 for WP2, as shown in table 16 in section 8, this work consists a pilot for adding support for other types of data, such as images and video, to Typhon polystores. To support different data modalities, we would need to analyse processing requirements, similarly to the analysis of text processing requirements, and develop a type system that allows for stand-off annotations, accordingly. A major part of the type system presented in section 3 would be reusable for other modalities. For example, the classification data type can be extended to different data modalities, to cover object recognition in images, voice identification in audio data, etc. Data types specific to text, such as sentences and tokens, will have to be replaced by data types that correspond to parts of a data instance of the target modality, e.g. frames and pixels in videos.

In the next section, we discuss extensions of the type system to cover specific requirements of each Typhon use case partner.

Table 7: Data types specific to the use case of Alpha Bank

Type	Text unit	Attributes
Account number	Word	Account Number + begin: Int + end: Int + accountNumber: str
Customer Unique Number	Word	Customer Unique Number + begin: Int + end: Int + customerUniqueNumber: str
Credit Amount	Word	Credit Amount + begin: Int + end: Int + creditAmount: Int
debit Amount	Word	Debit amount + begin: Int + end: Int + debitAmount: Int
Transaction Amount	Word	Transaction Amount + begin: Int + end: Int + transactionAmount: Int
Date transaction valid in bank	Word	Date transaction valid in bank + begin: Int + end: Int + dateTransactionValid: str
Card Number	Word	Card Number: Int + begin: Int + end: Int + cardNumber: Int

5 Use cases

In this section, we present different use cases provided by the Typhon use case partners, requiring the design and implementation of bespoke text data types to cover domain-specific requirements. Such specialised data types extend the general purpose Typhon text data type system presented in section 3. More specifically, Alpha bank requires modelling text data types relevant to bank customers while Volkswagen and GMV require identifying specialised products in their textual datasets. They also need to analyse customer feedback relevant to their products. Furthermore, Nea Odos requires identifying specialised named entities relevant to weather and traffic conditions.

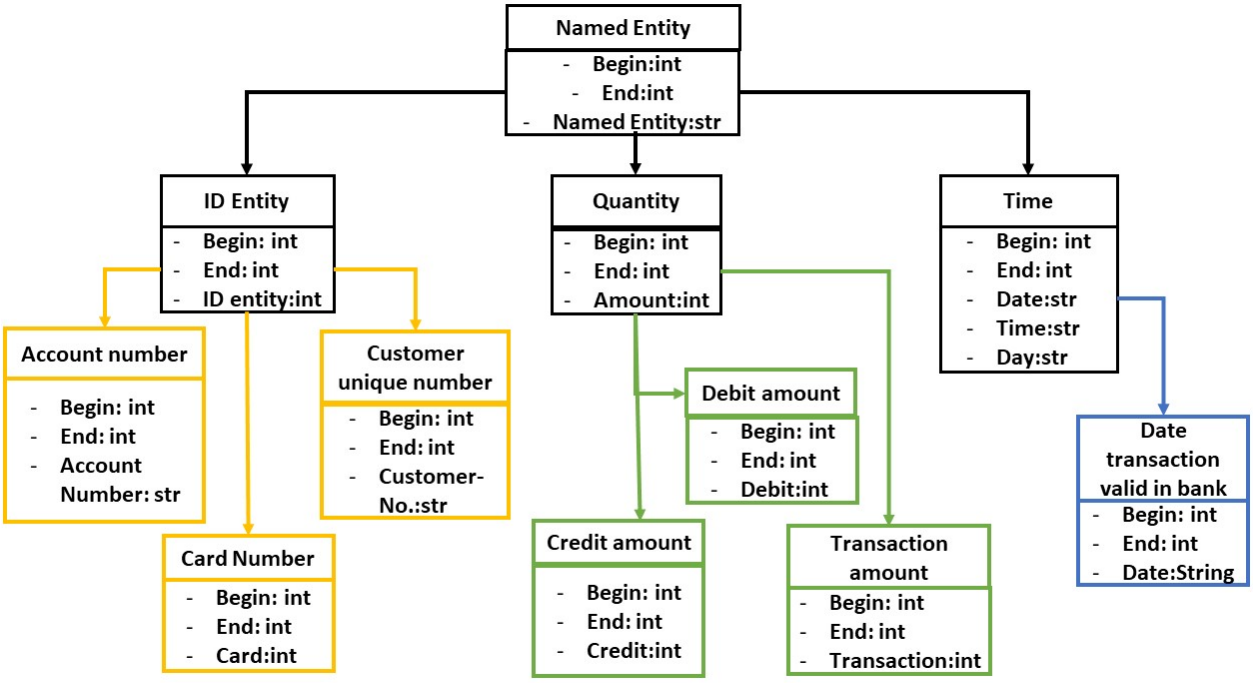

Figure 4: Alpha type-system

5.1 Alpha Bank

Alpha bank provided different instances of unstructured text that need to be structured using certain rules. Their text collections consist of customer letters requesting services such as taking loans, or opening\closing accounts and customer complaints or feedback. Alpha bank requested several specialised data types which are outlined in Table 5. Based upon this, we have extended our general purpose type system with these specialised data types. Figure 4 illustrates our extensions to the general purpose type system. Boxes in black boundaries represent general purpose data types, while boxes in coloured boundaries (yellow,green and blue) show the new data types created for this use case. *Account number*, *Customer unique number*, and *Card number* all extend the general purpose data type *ID*. Additionally, the *Credit amount*, *Debit amount* and *Transaction amount* inherit the general purpose data type *Quantity*. Lastly, we included a *Date transaction valid in bank* type which is a sub-type of the *Time* data type. These specialised data types can be used by any financial institution, if required in the future.

5.2 Volkswagen

VW required data types which are specific to their domain. Figure 5 and Table 8 illustrate these data types which extend our general purpose type system. The VW-specific data types include the *Car* type which extends the *Product* type and it consists of a *car make* and *car model* attributes.

VW also required modelling a *Sentiment Analysis* data type. More specifically, VW provided a sample dataset consisting of customer feedback relevant to their car rentals services. The customer feedback dataset is written in German. Based on this, we developed a Sentiment Analysis data type with attributes: begin/end reference pointer, polarityLabel (String - denotes the polarity label of a document, i.e. positive/neutral,negative)

Table 8: Data types specific to the use case of VW

Type	Text unit	Attributes
Car	Word/phrase	Car + begin: Int + end: Int + Car make: str + Car model
Car Year	Word	Car Year + begin: Int + end: Int + Car Year: str
Date of Purchase	Word/phrase	Date of Purchase + begin: Int + end: Int + Date of Purchase: str
Date of Guarantee	Word/phrase	Date of Gurantee + begin: Int + end: Int + Date of Gurantee: str
Sentiment Analysis	Word	Polarity + begin: Int + end: Int + polarity: str + polarityScore: Float

and polarityScore (Float - a confidence value which shows how likely it is for a document to belong to its corresponding polarity label).

In addition to the *Car* and *Sentiment Analysis* data types, we created three new sub-type extensions, namely *Car Year*, *Date of Purchase* and *Date of Guarantee*, of the general purpose data type *Time*. The *Car Year* shows the build date of a particular car, the *Date of Purchase* indicates the date when a customer bought a car and the *Date of Guarantee* is used to identify the types of car services that are available to a customer according to the guarantee start date.

5.3 GMV

Regarding GMV, we identified several domain-specific data types which are presented in Table 9 and in figure 6. The first domain-specific data type is relevant to *GMV Products*. A GMV Products consists of the following attributes: begin/end reference pointer, *productSector* (String - GMV Product sectors include: Aeronautics, Banking And Insurance, Cybersecurity, Defense And Security, Gnss, Healthcare, ICT For Business, Intelligent Transportation Systems, Public Administrations, Space, Telecommunications.) and *productMake* (String - actual name of a product).

Similarly to VW, GMV requires a *Sentiment Analysis* data type to analyse their product reviews datasets and to subsequently ensure the quality of their products and to better understand feedback received by their customers.

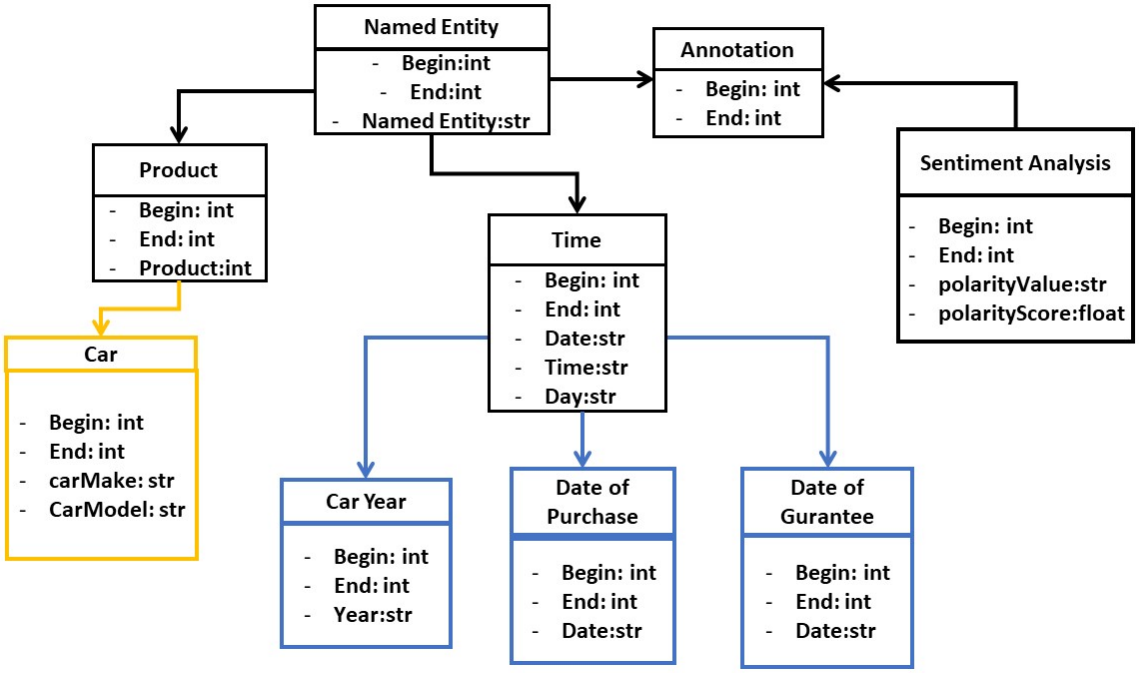

Figure 5: VW type-system

Considering that GMV products span across a wide range of different sectors, our sentiment analysis pipelines may have to be adapted to the terminology used in each specific domain/sector in order to more accurately analyse opinions expressed in text.

The last specialised data type that we developed is the *Date of Purchase* which extended the general purpose data type *Time*. This data type is required by GMV for monitoring their sales.

5.4 Nea Odos

Nea Odos requires modelling several specialised textual data types which are illustrated in Figure 7 and in Table 10. More specifically, they need to model traffic and weather conditions, and possible car collusion incidents to better manage the operation of the toll lanes. Time data types are also important for Nea Odos for the following reasons. Firstly, they need to identify the date and time when increased traffic appears. Secondly the time data type will allow Nea Odos to notify drivers via the VMSs (Variable Message Signs), which are installed in key points along their motorways, when extreme weather conditions will occur. Finally, Nea Odos required modelling the driving *licence number* and *video recordings* of an incident.

In this section, we have presented domain-specific extensions to our general purpose Typhon text data type system which are tailored to specific use case scenarios provided by the 4 Typhon use case partners. We have further demonstrated that our general purpose data type system can be readily extended with new types that model a wide range of different application requirements.

Table 9: Data types specific to the use case of GMV

Type	Text unit	Attributes
GMV Products	Word	GMV Product + begin: Int + end: Int + productSector: str + productMake: str
Date of Purchase	Word	Date of Purchase + begin: Int + end: Int + dateOfPurchase: str
Sentiment Analysis	Word	Polarity + begin: Int + end: Int + polarityLabel: str + polarityScore: Float

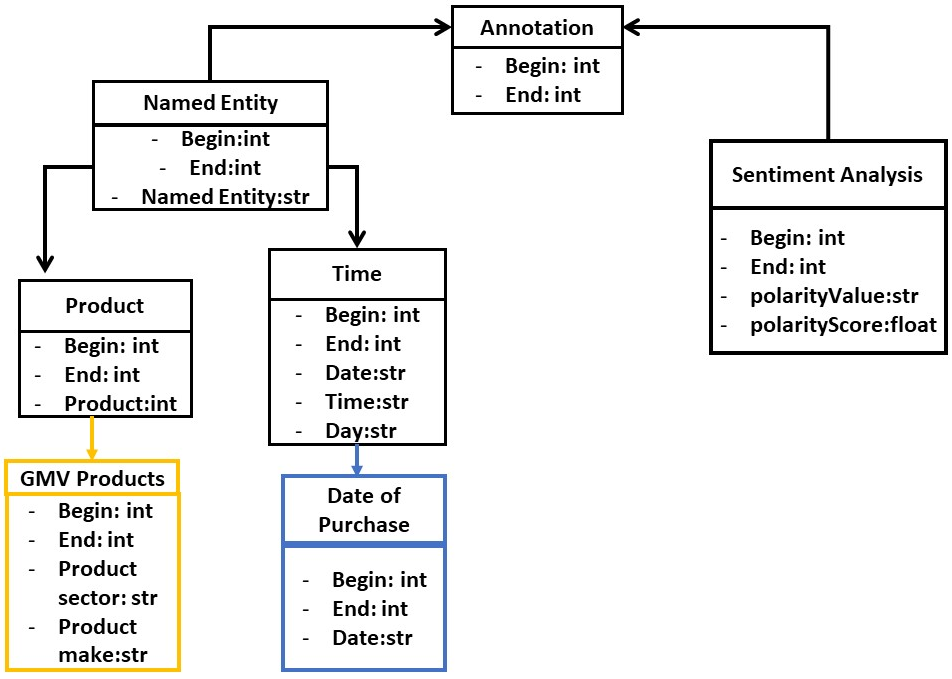

Figure 6: GMV type-system

Table 10: Data types specific to the use case of Nea Odos

Type	Text unit	Attributes
Licence number	Word	Licence Number + begin: Int + end: Int + licenceNumber: str
Date/Days	Word	Date/Days + begin: Int + end: Int + days: str
Weather Conditions	Word	Weather Conditions + begin: Int + end: Int + weatherConditions: str
Average Vehicle Speed	Word	Average Vehicle Speed + begin: Int + end: Int + averageVehicleSpeed: Int
Date	Word	Date + begin: Int + end: Int + date: Int

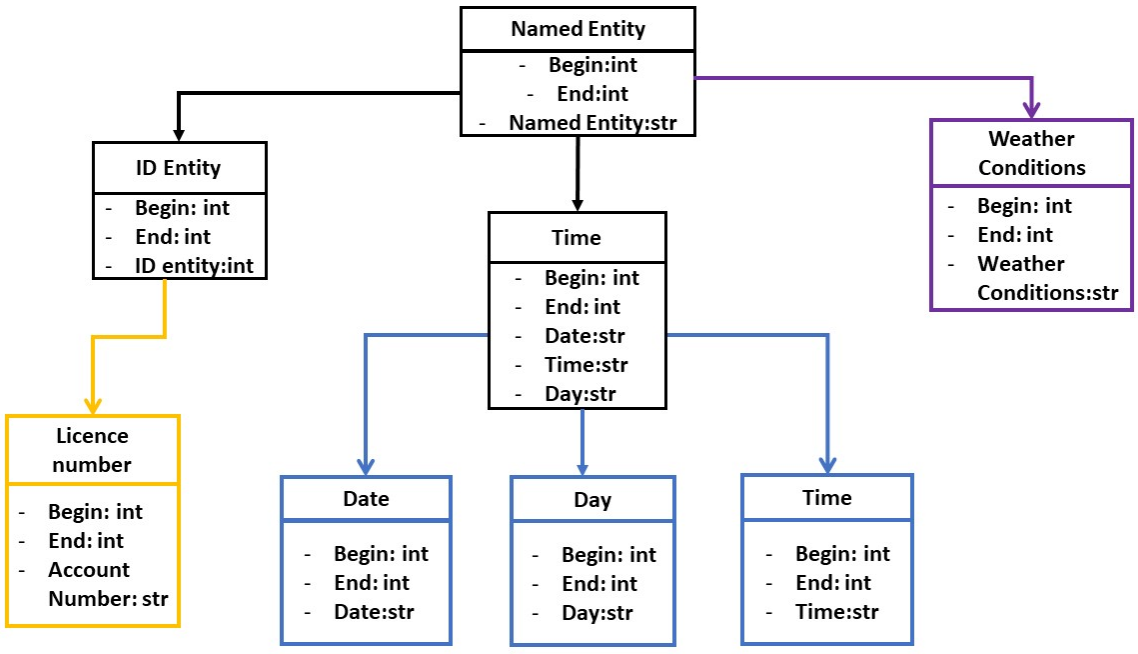

Figure 7: Neo Type-System

6 Sentiment Analysis: A use case application of the Typhon text data type system

In this section, we demonstrate how the proposed Typhon text data type system can be used for the development of a real-world text processing pipeline. More specifically, we implement a sentiment analysis UIMA pipeline that builds upon our type system to automatically annotate documents with sentiment labels. The developed UIMA pipeline co-ordinates supervised machine learning algorithms, available in the WEKA platform, that learn to discriminate between positive, negative and neutral documents. The machine learning algorithms require pre-processing the input documents using different text mining components, such as tokenisers, stemmers and n-gram extractors to more accurately identify sentiment labels in a document collection.

The section starts by firstly explaining the different data types of the Typhon system which are used in a sentiment analysis pipeline. It then presents an UIMA sentiment analysis pipeline which is evaluated against several publicly available datasets. Experiments conducted show that our Typhon data type system can aid rapid development of sentiment analysis pipelines wherein text processing components can be combined in a straightforward way using shared data types.

6.1 Sentiment Analysis Typhon Type System

Figure 8 shows the data types of the Typhon type system used in a sentiment analysis pipeline. The *Document* data type stores the *Language* and *InputFile* (i.e. file path) of an input document while the *Tokens* type models the constituent words of a document. The *Stem* data type converts the constituent words into their root forms while the *NGrams* type models Bigrams and Trigrams which are multi-word phrases consisting of two or three words, respectively. Finally the *Sentiment Analysis* data type stores the sentiment label of the document.

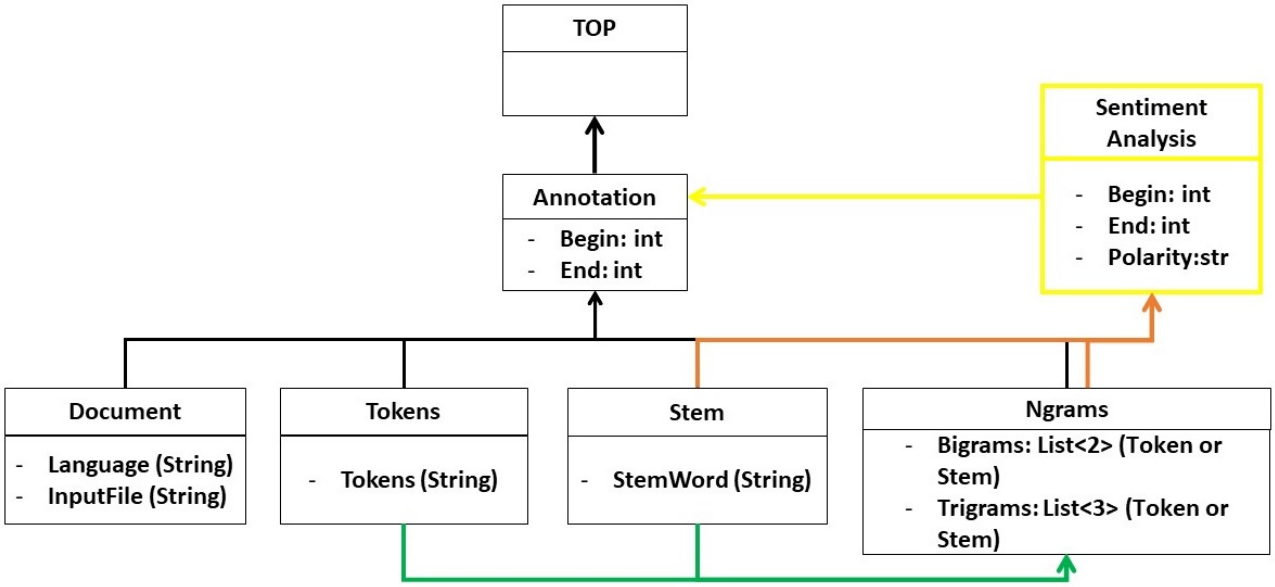

Figure 8: Typhon text data types used in sentiment analysis.

6.2 Sentiment Analysis UIMA pipeline

A UIMA pipeline implements three basic operations, namely read, process and write. The read operation is realised by a *Collection Reader* component which reads a document from a pre-specified location (e.g. file stored on disk, textual attribute of a Typhon polystore) and loads the content of the document into memory, the process operations is implemented by an *Analysis Engine* which is essential a text mining component(s) that processes the input document while the write operation is implemented by a *CAS Consumer* which stores the text mining analysis results into an pre-specified output location. Any UIMA pipeline should consist of one Collection Reader (read operation) any number of Analysis Engines (process operation) and finally one CAS Consumer (write operation).

Figure 9 shows our proposed sentiment analysis UIMA pipeline. The pipeline firstly uses a Collection Reader (green box of the figure) to load the document collection and to generate *Document* data types (one for each document of the collection). Following the Collection reader, we co-ordinate three Analysis Engines (bright orange boxes), namely a Tokeniser, a Stemmer and NGrams. The different data types generated by the three Analysis Engines are illustrated in orange-dashed boxes. It should also be noted that three Analysis Engines of our pipeline can be implemented using different libraries. As an example, a Tokeniser can be implemented using either standard string processing functions in Java (e.g. splitting by white space character) or using the LingPipe toolkit. We thus experiment with different combinations of available libraries in order to identify optimal configuration of our sentiment analysis pipeline. The yellow boarded boxes (See figure 9) show the various libraries we used for the same component (i.e. same data type). Finally, the *WEKA Train* is a CAS Consumer component which uses the WEKA machine learning platform to train a machine learning algorithm and to and subsequently produce sentiment annotations (i.e. polarity label of each document).

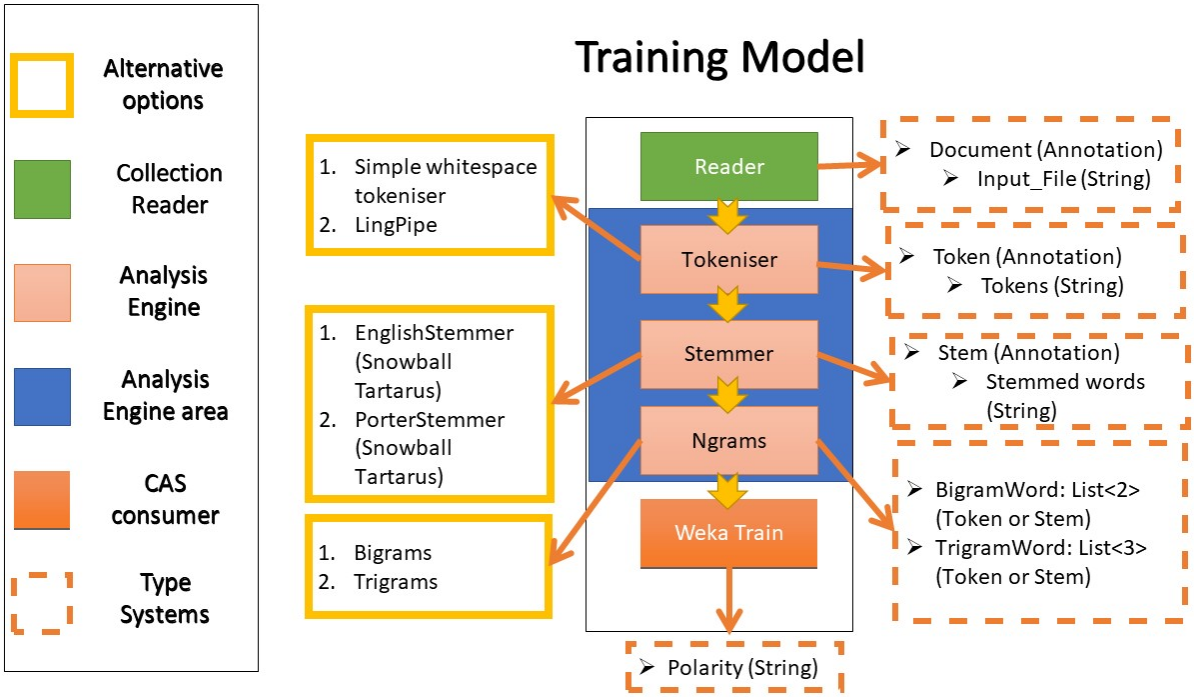

Figure 9: A UIMA sentiment analysis pipeline that uses the proposed Typhon text data type system to automatically annotate documents with sentiment labels.

6.3 Evaluation of the Sentiment Analysis pipeline

We have conducted 6 large-scale experiments to investigate the optimal configuration of our sentiment analysis pipeline. More specifically, we evaluate the performance of different pre-processing components (e.g. Stemmer implemented either in CoreNLP or in Snowball Tartarus), feature weighting schemes (either TF or TF-IDF), feature types (e.g. bigrams, trigrams, bigrams+trigrams), feature filtering (i.e. removes features according to a predefined frequency threshold) and classification algorithms (e.g. Support Vector Machines, Random Forest, Naive Bayes) available in the WEKA platform. It should be noted that the different pipeline configurations were automatically created simply by changing the XML descriptor file of the sentiment analysis pipeline.

The different pipeline configurations are evaluated in terms of accuracy, precision, recall and F-score using 10-fold cross validation. More formally, given TP be the number of True Positives, TN for True Negatives, FP for False Positives and FN for False Negatives the 4 evaluation metrics are computed as follows:

$$\text{precision} = \frac{TP}{TP + FP}, \text{ recall} = \frac{TP}{TP + FN}, \text{ F-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

In addition to the 4 evaluation metrics, we also computed the execution time of each pipeline configuration.

We used 6 publicly available datasets to assess the performance of the sentiment analysis pipelines. Table 11 summarises various characteristics of the datasets including:

1. the source and name of the dataset
2. the size in terms of number of documents
3. the number of documents labelled as positive
4. number of documents labelled as negative
5. number of neutral documents (this label is only available for the SemEval117 dataset)

Table 11: Data Sources

Dataset	**size**	**positive**	**negative**	**neutral**
UMICH SI650 [82]	7,086	3,995	3,091	...
Amazon cell [42]	1,000	500	500	...
Yelp [42]	1,000	500	500	...
Imbd [42]	1,000	500	500	...
Sentiment140 [53]	1,048,575	554,470	494,105	...
SemEval-17 [73]	20,632	7059	3,231	10,342
Sum	1,079,293	567,024	501,927	10,342

For each of the datasets, we identified the best pipeline configuration according to the F-score performance and the execution time.

6.3.1 Performance of pre-processing components

The first experiment evaluates the performance of our UIMA sentiment analysis pipeline when using different combinations of pre-processing components. We build upon the UIMA framework to plug and play pre-processing components into pipelines making it easier to identify the best pipeline configuration. We develop 4 pipeline configurations using a combination of 2 Tokenisers and 2 Stemmers:

1. Standard tokeniser (T1): our own tokeniser which simply segments a document into its constituent using the whitespace delimiter character.
2. StringTokenizer (T2): from the java util package[8]
3. englishStemmer (S1): from the tartarus snowball package[9]
4. PorterStemmer (S2): from the tartarus snowball package[10]

Table 12 shows the average performance of the 4 pipeline configurations when applied to the 6 evaluation datasets. The performance is computed in terms of accuracy, precision, recall and f-score while the reported results are average values across the performance obtained by 5 classification algorithms, namely Naive Bayes, Random Forest, SVM, Complement Naive Bayes and LibLinear. It can be observed that the T1-S1 pipeline configuration obtained the best results in most cases although the performance improvements over the remaining three configurations were insignificant.

6.3.2 Performance of classification algorithms

In this section, we report the performance obtained by 5 classification algorithms available in WEKA, i.e. namely Naive Bayes (NB), Random Forest (RF), SVM, Complement Naive Bayes (CNB) and LibLinear (LIB), when using the 4 different combinations of pre-processing components. We developed a total number of 20 pipeline configurations (i.e. 5 classifier$\times$4 pre-processing components). In addition to the F-score performance, we also report the time needed to execute each pipeline configuration.

Table 13 shows the highest and lowest F-Score and the slowest and fastest execution time achieved by the different pipeline configurations. We further report the best pipeline configuration when taking into consideration both the F-Score performance and the execution time. As an example, regarding the $UMICH$ dataset, we observe that the SVM-T2-S1 pipeline obtains an F-Score performance of 0.991 which is only marginally lower than the F-Score achieved by the RF-T2-S1 pipeline (highest F-Score performance of 0.998). However, the SVM-T2-S1 pipeline, which is our preferred configuration, is substantially faster than the RF-T2-S1 pipeline.

Overall, the CNB classifier obtained both a high F-score performance and a fast execution time in 5 out of 6 datasets.

6.3.3 Performance of feature weighting schemes

Table 14 shows the accuracy, precision, recall and F-score performance of the CNB-T1-S1 configuration pipeline, which obtained a robust performance and fast execution time in the previous experiments, when

[8] docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html
[9] snowball.tartarus.org/algorithms/english/stemmer.html
[10] snowball.tartarus.org/algorithms/porter/stemmer.html

Table 12: Average performance results of our sentiment analysis pipeline using 4 different combinations of pre-processing components, i.e. T1-S1, T1-S2, T2-S1. The results are average values of the accuracy, precision, recall and f-score performance obtained by 5 classification algorithms, namely Naive Bayes, Random Forest, SVM, Complement Naive Bayes and LibLinear

		Pipeline Configuration			
	Performance	**T1- S1**	**T1-S2**	**T2-S1**	**T2-S2**
UMICH	**Accuracy**	**.964**	.959	.954	.955
	Precision	**.968**	.967	.962	.962
	Recall	.967	.958	.957	**.969**
	F-score	**.964**	.958	.953	.955
Amazon	**Accuracy**	.802	**.803**	.802	**.803**
	Precision	.807	.807	.807	.807
	Recall	.802	**.803**	.802	**.803**
	F-score	.801	**.802**	.801	**.802**
Imdb	**Accuracy**	**.736**	.734	.719	.717
	Precision	**.740**	.738	.725	.723
	Recall	**.740**	.733	.719	.717
	F-score	**.734**	.732	.717	.715
Yelp	**Accuracy**	**.765**	.764	.745	.750
	Precision	**.770**	.769	.750	.755
	Recall	**.770**	.764	.745	.750
	F-score	**.770**	.764	.744	.749
Senti-140	**Accuracy**	**.786**	.785	.783	.783
	Precision	**.789**	**.789**	.787	.787
	Recall	**.788**	.787	.785	.784
	F-score	**.785**	**.785**	.782	.782
SemEval	**Accuracy**	**.791**	.789	.774	.777
	Precision	**.789**	.785	.774	.779
	Recall	**.760**	.758	.746	.745
	F-score	**.757**	.756	.740	.741

using the TF and TF-IDF feature weighting schemes. Here, we note that the TF weighting scheme achieved a slightly higher classification performance than the TF-IDF scheme in 4 out of 6 datasets. Regarding the execution time, the TF-IDF scheme was faster than the TF scheme in 5 out of 6 datasets. A larger and substantial time execution margin of approximately 9 seconds is observed on the Senti-140 dataset.

6.3.4 Performance of feature types

Feature engineering, i.e. selecting appropriate features types (e.g. unigrams, bigrams, trigrams or a combination of the above) for training a machine learning classifier, is prominent in sentiment analysis and can substantially affect the performance of the classifier[76, 62].

The most commonly used features in sentiment analysis are n-grams, pos tags and lexicon derived features [2, 23]. In the context of this study we only investigate the performance of different combinations of n-grams.

As future work we plan to improve upon the performance of our sentiment analysis pipeline by including additional feature types.

Table 15 shows the average performance of the following n-gram combinations: 1) Unigrams alone, 2) Bigrams alone, 3) Trigrams alone, 4) Unigrams and Bigrams, 5) Unigrams and Trigrams, 6) Bigrams and Trigrams, 7) Unigrams Bigrams and Trigrams and 8) All Ngrams combined. The average performance is computed across all pipeline configurations (i.e. different pre-processing components and classification algorithms).

The obtained results show that the trigram features yielded the lowest performance in most cases while a combination of all n-grams led to the highest performance in 3 out of the 6 datasets. Unigrams and trigrams combined obtained the highest performance in the Yelp dataset. We further observe that the performance margin between the different feature types is substantially high in several occasions. As an example, the unigrams features showed a performance improvement in terms of F-score of 27.6% over the trigram features on the Imdb dataset. This determines that a careful feature selection method is required in order to optimise the performance of sentiment analysis pipelines.

6.3.5 Performance of feature filtering

The last experiment investigates the performance of our sentiment analysis pipeline when using a frequency threshold filtering method. Considering that one of the research objectives of this project is to scale the text processing pipelines to big data collections, we are interested in reducing the computational resources needed to execute the UIMA pipelines without reducing the accuracy of the underlying text mining models. Feature filtering techniques can decrease the memory requirements of machine learning classifier by eliminating non-informative features [89].

We implement a frequency threshold method that removes features (i.e. n-grams) with a frequency (i.e. number of times an n-gram occurs in the document collection) less than a pre-defined threshold. Figure 10 shows the F-Score performance of our sentiment analysis pipeline when using a varying frequency threshold between $[1, 30]$. Here, we observe that for datasets that are smaller in size, the performance of the sentiment analysis pipeline decreases as we increase the frequency threshold. As an example, the F-score performance on the Amazon dataset, which consists of only $1,000$ documents, substantially drops from 0.832 for a frequency threshold of 1 to 0.676 for a threshold of 30. However, for larger dataset (e.g. Senti-140 that contains more than 1M documents), the F-Score performance shows insignificant variations as we increase the frequency threshold.

6.4 Summary

We have developed and evaluated a UIMA-based sentiment analysis pipeline that uses our proposed Typhon text data type system to annotate several datasets with relevant text types. We have further demonstrated that the Typhon type system accelerates the development process of text processing pipelines considering that heterogeneous text mining components, which are implemented by different software libraries, can be freely combined into unified pipelined applications. We have experimented with a wide range of different pipeline configurations, such as different combinations of pre-processing components and classification algorithms, feature weighting schemes, feature types and feature filtering, in order to identify the optimal parameter settings for our sentiment analysis pipeline.

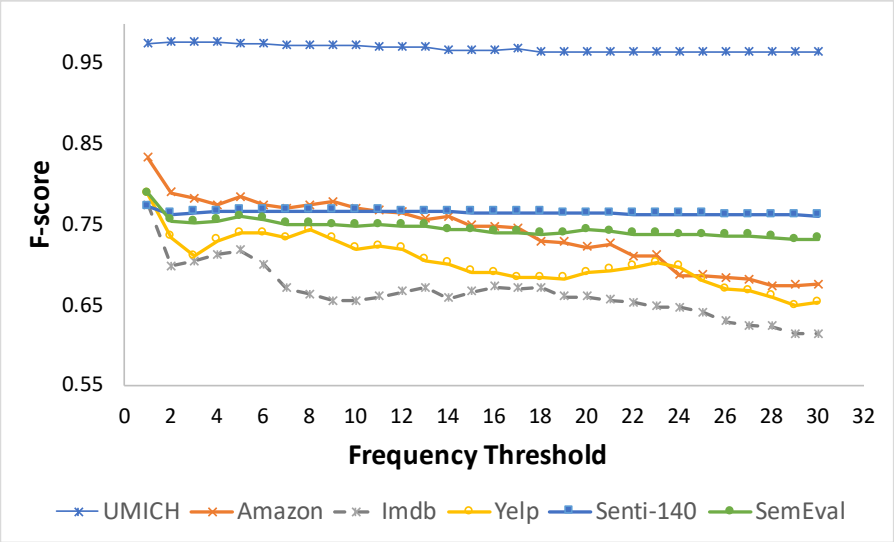

Figure 10: F-Score performance of the sentiment analysis pipeline on an increasing frequency threshold of features that are retained in the dataset.

Table 13: Best pipeline configurations in terms of both F-Score and execution time across the 6 evaluation datasets. The table also reports the highest and lowest F-Score and the slowest and fastest execution time obtained by the different pipeline configurations.

		Pipeline Configuration	F-score	Time (mm:ss)
UMICH	Highest F-score	RF-T2-S1	.998	17:46
	Lowest F-Score	NB-T2-S1	.807	04:11
	Slowest Time	RF-T1-S2	.997	22:23
	Fastest Time	CNB-T2-S2	.979	00:01
	Best Configuration	**SVM-T2-S1**	**.991**	**00:05**
Amazon	Highest F-Score	CNB-T1-S1	.831	00:01
	Lowest F-Score	NB-T1-S1	.748	00:05
	Slowest Time	RF-T1-S2	.777	06:09
	Fastest Time	CNB-T2-S2	.829	00:01
	Best Configuration	**CNB-T1-S1**	**.831**	**00:01**
Imbd	Highest F-Score	CNB-T1-S1	.787	00:02
	Lowest F-Score	NB-T2-S1	.675	01:00
	Slowest Time	RF-T2-S1	.699	05:13
	Fastest Time	CNB-T1-S2	.773	00:02
	Best Configuration	**CNB-T1-S1**	**.787**	**00:02**
Yelp	Highest F-Score	CNB-T1-S2	.798	00:01
	Lowest F-Score	NB-T2-S1	.665	01:02
	Slowest Time	RF-T2-S2	.745	04:35
	Fastest Time	CNB-T1-S1	.784	00:00
	Best Configuration	**CNB-T1-S2**	**.798**	**00:01**
SemEval	Highest F-Score	CNB-T1-S1	.832	00:02
	Lowest F-Score	NB-T2-S2	.588	07:06
	Slowest Time	RF-T1-S1	.753	01:22:05
	Fastest Time	CNB-T1-S1	.808	00:01
	Best Configuration	**CNB-T1-S1**	**.832**	**00:02**
Senti-140	Highest F-Score	LIB-T1-S1	.798	02:03:42
	Lowest F-Score	CNB-T2-S1	.768	00:27:04
	Slowest Time	LIB-T2-S2	.796	02:07:00
	Fastest Time	CNB-T1-S2	.779	00:26:20
	Best Configuration	**CNB-T1-S2**	**.779**	**00:25**

Table 14: Performance of the CNB-T1-S1 configuration pipeline when using the TF and TF-IDF feature weighting schemes across the 6 dataset. The table also records the execution time of the two weighting schemes.

		TF	TF-IDF
UMICH	**Time (sec)**	**.055**	.577
	Accuracy	**.982**	.974
	Precision	**.982**	.973
	Recall	**.981**	.975
	Fscore	**.982**	.974
Imdb	**Time (sec)**	.067	**.064**
	Accuracy	**.782**	.774
	Precision	**.791**	.778
	Recall	**.782**	.775
	Fscore	**.780**	.774
Senti-140	**Time (sec)**	34.277	**25.061**
	Accuracy	.772	.772
	Precision	.780	.780
	Recall	.777	.777
	Fscore	.772	0.772

		TF	TF-IDF
Amazon	**Time (sec)**	.022	**.018**
	Accuracy	**.835**	.833
	Precision	**.839**	.835
	Recall	**.835**	.833
	Fscore	**.834**	.833
Yelp	**Time (sec)**	.387	**.02**
	Accuracy	.787	**.788**
	Precision	**.791**	.790
	Recall	**.787**	.788
	Fscore	.786	**.788**
SemEval	**Time (sec)**	1.119	**.278**
	Accuracy	**.839**	.810
	Precision	**.801**	.780
	Recall	**.815**	.806
	Fscore	**.807**	.789

Table 15: Features:This table represents an average of all the models including all datasets, preprocessing techniques and classifiers for each of the features.

	Dataset	UMICH	Amazon	Imdb	Yelp	Senti-140	SemEval
Unigrams	**Accuracy**	.930	.816	**.816**	.797	.738	.837
	Precision	.972	.819	**.816**	.819	.744	.827
	Recall	.973	.816	**.816**	.797	.743	.782
	Fscore	.972	.816	**.816**	.797	.739	.798
Bigrams	**Accuracy**	.952	.704	.635	.680	.732	.773
	Precision	.950	.727	.637	.688	.740	.746
	Recall	.955	.704	.635	.680	.737	.775
	Fscore	.952	.696	.634	.677	.732	.753
Trigrams	**Accuracy**	.968	.608	.575	.691	.696	.512
	Precision	.966	.676	.607	.663	.714	.664
	Recall	.970	.608	.575	.601	.703	.638
	Fscore	.968	.567	.540	.560	.694	.518
Unigrams and Trigrams	**Accuracy**	.969	.702	.648	.681	.749	.681
	Precision	.967	.728	.650	.691	.750	.707
	Recall	.971	.702	.648	.681	.745	.736
	Fscore	.969	.693	.647	.677	.740	.676
Unigrams and Bigrams	**Accuracy**	.975	**.835**	.718	.796	.768	.840
	Precision	.973	**.837**	.786	.798	.774	.819
	Recall	.975	**.835**	.781	.796	.772	.802
	Fscore	.974	**.835**	.780	.796	.768	**.809**
Unigrams and Trigrams	**Accuracy**	.978	.831	.807	**.800**	**.877**	.821
	Precision	.977	.833	.810	**.801**	.774	.793
	Recall	.979	.831	.807	**.800**	.771	**.815**
	Fscore	.978	.831	.807	**.800**	.766	.801
All Ngrams	**Accuracy**	.980	.831	.788	.794	.772	**.844**
	Precision	**.980**	.833	.792	.796	**.780**	**.832**
	Recall	**.979**	.831	.788	.794	**.777**	.793
	Fscore	**.980**	.831	.787	.794	**.772**	.808

7 Risks

Identified risks are mainly about the possible difficulties in implementing the text processing tasks[11], described by our type system, in the future. The main sources of potential difficulties are summarised below:

1. lack of adequate amounts of manually labelled data required to train machine learning algorithms
2. ambiguous terms and phrases. for example, “Asia” and “Jordan” are names of locations and also names of people.
3. lack of text processing components in under-resourced languages. For example, in Greek there may be difficulties in finding readily available text processing tools and libraries.
4. lack of full access to the data of Typhon use case partners. For example, parts of Alpha Bank’s data cannot be shared outside the bank’s premises. As a result, we have to train our models on data similar to the real one and then share our trained workflows with Alpha Bank for local deployment. In this case, fine-tuning the parameters of the system may not be ideal.

Additionally, risks include the add-on delay when running application through UIMA. Processing time may be significantly more than existing text processing components, e.g. Weka, without the UIMA wrapper.

[11] All text processing tasks associated with the corresponding data types in the type system are shown in Appendix A.

8 Typhon requirements

In this section, we review the technology requirements and use case requirements of Deliverable 1.1 that are related to text processing. Tables 16 and 17 present our progress towards each technology and use case requirement, respectively. Requirements that are to be completed are marked with the task number and the time frame in which they will be addressed.

Table 16: Consolidated Technology Requirements

Number	Requirement	Priority	Status	Future task	Time frame (M)
WORKPACKAGE 2: HYBRID POLYSTORE DESIGN					
D1	TyphonML shall enable the specification of data entities and relationships that will be stored in TyphonML	Shall	Done		
D4	Definition of custom data types to be used in TyphonML data models shall be supported.	Shall	Done		
D5	Specification of data types that are needed for applying text-specific analysis (e.g. text, video, recordings) shall be supported.	Shall	Done		
D6	The definition of structured data types (e.g. sentences, facts, entities, events) that can be extracted from text and represented in TyphonML shall be supported.	Shall	Done		
WORKPACKAGE 4: HYBRID POLYSTORE QUERYING					
D33	The TyphonQL engine shall support normalization of natural language fragments to enable "querying modulo spelling".	Shall	To be done	5.4	12-30
D36	TyphonQL shall support querying textual data.	Shall	To be done	5.4	12-30
WORKPACKAGE 5: HYBRID POLYSTORE ANALYTICS AND MONITORING					
D50	The development of text mining pipelines for data events shall be simplified.	Shall	To be done	5.4	12-30
WORK PACKAGE 7: PLATFORM INTEGRATION AND EVALUATION					
D76	Each of the Typhon components shall adhere to the specified TYPHON architectural guidelines	Shall	Ongoing		
D77	Each of the Typhon components shall use Git for source code control	Shall	Ongoing		

Table 17: Use-case requirements

Number	Requirement	Priority	Status	Future task	Time frame (M)
TEXT DATA MODELLING					
27	The text data storage shall be able to parse its data to a relational database	Shall	Done		
28	The text data storage shall be able to parse its data to an array database	Shall	Done		
29	Text data modelling for XML files shall be provided	Shall	Done		
POLYSTORE QUERY LANGUAGE					
47	The polystore query language should expose the same semantic data types and operations of the underlying database technologies as defined in their schemas or metamodels	Should	Done		
52	The query language shall be able to interpret and execute text search queries	Shall	To be done	5.4	12-30
QUERIES ON STRUCTURED DATA					
58	The system shall be able to process queries on relational databases	Shall	To be done	5.4	12-30
61	The system shall be able to process queries on text stores	Shall	To be done	5.4	12-30
QUERIES ON TEXTUAL DATA					
63	The polystore query language should expose a relevant subset of the data types and operations of at least one of the following: Solr, Lucene	Should	To be done	5.4	12-30
64	In the text data queries it shall be possible to search for one or more different keywords in one query	Shall	To be done	5.4	12-30
65	Text data queries using patterns or full text search shall be supported	Shall	To be done	5.4	12-30
66	The text data queries may be able to autocorrect words	May	To be done	5.4	12-30
67	The text data queries may be able to recognize incorrect spellings and mark them	May	To be done	5.4	12-30
68	The text data queries may be able to recognise different spellings for one word (AE/BE)	May	To be done	5.4	12-30
69	The text data queries shall be able to recognize the Greek language	Shall	To be done	5.4	12-30

9 Conclusion

In this deliverable, we presented a text data type system for modelling annotations that are automatically extracted from natural language text. Previous literature related to types of text annotations, platforms supporting text annotations and type-systems was investigated. The results of our literature review showed that stand-off annotations better satisfy the requirements of our project when compared to in-line annotations. We further reviewed existing text modelling and text processing platforms and found that although all the platforms had some advantages in text processing, the best platform to fit the needs of the project is the Unstructured Information Management Architecture (UIMA), due to its flexibility and extensibility.

Based on the general text processing requirements of Typhon use case partners, we developed a type system that addressed them. Then, we commented on the flexibility of the type system and illustrated how it can be extended. In particular, we extended our general type system with data types related to specific applications of interest to Typhon use case partners.

To demonstrate the use of the type system in practice, we developed and evaluated sentiment analysis workflows. The results show that our type system can help in developing sentiment analysis pipelines rapidly. Additionally, the type system allows to synthesise into workflows different text processing components easily, using shared data types. Based on the results of this work, we are preparing a paper that is currently at a draft stage and will be submitted in the near future.

In Task 5.4 (Month 12-30), we will use these data-types as a base to create text processing pipelines. Additionally, we will design and develop text processing components that help in faceted search, in identifying new information (e.g. categories or classes) and in applying topic modelling and text classification.

References

[1] Fabian Abel, Ilknur Celik, Geert-Jan Houben, and Patrick Siehndel. Leveraging the semantics of tweets for adaptive faceted search on twitter. In *International Semantic Web Conference*, pages 1–17. Springer, 2011.

[2] Nabeela Altrabsheh, Mihaela Cocea, and Sanaz Fallahkhair. Sentiment analysis: towards a tool for analysing real-time students feedback. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 419–423. IEEE, 2014.

[3] Vimala Balakrishnan and Ethel Lloyd-Yemoh. Stemming and lemmatization: A comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2(3), 2014.

[4] Mathias Bank and Martin Schierle. A survey of text mining architectures and the uima standard. In *LREC*, pages 3479–3486, 2012.

[5] Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. Developing a large semantically annotated corpus. In *LREC 2012, Eighth International Conference on Language Resources and Evaluation*, 2012.

[6] Riza Batista-Navarro, Jacob Carter, and Sophia Ananiadou. Argo: enabling the development of bespoke workflows and services for disease annotation. *Database*, 2016, 2016.

[7] Steven Bethard, Philip Ogren, and Lee Becker. Cleartk 2.0: Design patterns for machine learning in uima. In *International Conference on Language Resources & Evaluation:[proceedings]. International Conference on Language Resources and Evaluation*, volume 2014, page 3289. NIH Public Access, 2014.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[9] Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Angus Roberts, Valentin Tablan, Niraj Aswani, and Genevieve Gorrell. Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029, 2013.

[10] Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Angus Roberts, Valentin Tablan, Niraj Aswani, and Genevieve Gorrell. Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029, 2013.

[11] Kalina Bontcheva, Marin Dimitrov, Diana Maynard, Valentin Tablan, and Hamish Cunningham. Shallow methods for named entity coreference resolution. In *Chaınes de références et résolveurs d'anaphores, workshop TALN*, 2002.

[12] Jean-Sébastien Brunner and Thibaud Latour. Referencing text documents in multidimensional concept spaces for technological and scientific watch. In *Proceedings of" Workshop on Terminology, Ontology, and Knowledge Representation*, pages 22–23, 2004.

[13] Paul Buitelaar, Philipp Cimiano, Stefania Racioppa, and Melanie Siegel. Ontology-based information extraction with soba. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

[14] Johan Carlberger, Hercules Dalianis, Martin Duneld, and Ola Knutsson. Improving precision in information retrieval for swedish using stemming. In *Proceedings of the 13th Nordic Conference of Computational Linguistics (NODALIDA 2001)*, 2001.

[15] Damien Cram and Béatrice Daille. Terminology extraction with term variant detection. *Proceedings of ACL-2016 System Demonstrations*, pages 13–18, 2016.

[16] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[17] David S Day, Chad McHenry, Robyn Kozierok, and Laurel D Riek. Callisto: A configurable annotation workbench. In *LREC*, 2004.

[18] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[19] Manaal Faruqui, Sebastian Padó, and Maschinelle Sprachverarbeitung. Training and evaluating a german named entity recognizer with semantic generalization. In *KONVENS*, pages 129–133, 2010.

[20] David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[21] MAchine Learning for LanguagE Toolkit. MALLET website. `http://mallet.cs.umass.edu/`, 2018. Accessed: 2018-12-20.

[22] Maria Fuentes and Horacio Rodríguez. Using cohesive properties of text for automatic summarization. *JOTRI'02*, 2002.

[23] Alec Go, Lei Huang, and Richa Bhayani. Twitter sentiment analysis. CS224N Project Report, Stanford, 2009.

[24] Ralph Grishman. The nyu system for muc-6 or where's the syntax? In *Proceedings of the 6th conference on Message understanding*, pages 167–175. Association for Computational Linguistics, 1995.

[25] Stanford NLP Group. Standford-co-reference. `https://stanfordnlp.github.io/CoreNLP/coref.html`, 2018. Accessed: 2018-12-20.

[26] Emma Haddi, Xiaohui Liu, and Yong Shi. The role of text pre-processing in sentiment analysis. *Procedia Computer Science*, 17:26–32, 2013.

[27] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. Faceted wikipedia search. In *International Conference on Business Information Systems*, pages 1–11. Springer, 2010.

[28] Udo Hahn, Ekaterina Buyko, Rico Landefeld, Matthias Mühlhausen, Michael Poprat, Katrin Tomanek, and Joachim Wermter. An overview of jcore, the julie lab uima component repository. In *Proceedings of the LREC*, volume 8, pages 1–7, 2008.

[29] Udo Hahn, Ekaterina Buyko, Katrin Tomanek, Scott Piao, John McNaught, Yoshimasa Tsuruoka, and Sophia Ananiadou. An annotation type system for a data-driven nlp pipeline. In *Proceedings of the Linguistic Annotation Workshop*, pages 33–40. Association for Computational Linguistics, 2007.

[30] Martin Hassel. *Evaluation of automatic text summarizaiton: a practical implementation*. PhD thesis, Numerisk analys och datalogi, 2004.

[31] Nancy Ide, Christian Chiarcos, Manfred Stede, and Steve Cassidy. Designing annotation schemes: from model to representation. In *Handbook of Linguistic Annotation*, pages 73–111. Springer, 2017.

[32] Nancy Ide, Christiane Fellbaum, Collin Baker, and Rebecca Passonneau. The manually annotated sub-corpus: A community resource for and by the people. In *Proceedings of the ACL 2010 conference short papers*, pages 68–73. Association for Computational Linguistics, 2010.

[33] Radu Ion, Elena Irimia, Dan Stefanescu, and Dan Tufis. Rombac: The romanian balanced annotated corpus. In *LREC*, pages 339–344. Citeseer, 2012.

[34] Chuleerat Jaruskulchai and Canasai Kruengkrai. A practical text summarizer by paragraph extraction for thai. In *Proceedings of the sixth international workshop on Information retrieval with Asian languages-Volume 11*, pages 9–16. Association for Computational Linguistics, 2003.

[35] Ning Kang, Erik M van Mulligen, and Jan A Kors. Comparing and combining chunkers of biomedical text. *Journal of biomedical informatics*, 44(2):354–360, 2011.

[36] Ioannis Manousos Katakis, Georgios Petasis, and Vangelis Karkaletsis. Clarin-el web-based annotation tool. In *LREC*, 2016.

[37] Gitansh Khirbat. *Supervised Algorithms for Complex Relation Extraction*. PhD thesis, UNIVERSITY OF MELBOURNE, 2017.

[38] Youngjun Kim, Ellen Riloff, and Stéphane M Meystre. Improving classification of medical assertions in clinical notes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 311–316. Association for Computational Linguistics, 2011.

[39] BalaKrishna Kolluru, Lezan Hawizy, Peter Murray-Rust, Junichi Tsujii, and Sophia Ananiadou. Using workflows to explore and optimise named entity recognition for chemistry. *PloS one*, 6(5):e20181, 2011.

[40] Georgios Kontonatsios, Paul Thompson, Riza Theresa Batista-Navarro, Claudiu Mihăilă, Ioannis Korkontzelos, and Sophia Ananiadou. Extending an interoperable platform to facilitate the creation of multilingual and multimodal nlp applications. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 43–48, 2013.

[41] Tuomo Korenius, Jorma Laurikkala, Kalervo Järvelin, and Martti Juhola. Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 625–633. ACM, 2004.

[42] Dimitrios Kotzias, Misha Denil, Nando De Freitas, and Padhraic Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 597–606. ACM, 2015.

[43] Lun-Wei Ku, Li-Ying Lee, Tung-Ho Wu, and Hsin-Hsi Chen. Major topic detection and its application to opinion summarization. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 627–628. ACM, 2005.

[44] Fotis Lazarinis. *Text Extraction and Web Searching in a Non-Latin Language*. PhD thesis, University of Sunderland, 2008.

[45] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the fifteenth conference on computational natural language learning: Shared task*, pages 28–34. Association for Computational Linguistics, 2011.

[46] Elizabeth D Liddy and Sung-Hyon Myaeng. Tipster panel-dr-link's linguistic-conceptual approach to document detection. In *TREC*, pages 113–130, 1992.

[47] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining text data*, pages 415–463. Springer, 2012.

[48] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[49] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[50] Scott Mardis, John Burger, P Anand, D Anderson, J Griffith, M Light, C McHenry, A Morgan, and J Ponte. Qanda and the catalyst architecture. In *AAAI Spring Symposium on Mining Answers from Text and Knowledge Bases*, 2002.

[51] Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 92–97, 2013.

[52] Mohamed Mejri and Jalel AKAICHI. A survey of textual event extraction from social networks. *Proceedings of the First Conference on Language Processing and Knowledge Management*, 2017.

[53] Saif M Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. *arXiv preprint arXiv:1308.6242*, 2013.

[54] Thomas S Morton. Using coreference for question answering. In *Proceedings of the Workshop on Coreference and its Applications*, pages 85–89. Association for Computational Linguistics, 1999.

[55] Tony Mullen and Robert Malouf. A preliminary investigation into sentiment analysis of informal political discourse. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pages 159–162, 2006.

[56] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[57] Adeline Nazarenko, Erick Alphonse, Julien Derivière, Thierry Hamon, Guillaume Vauvert, and Davy Weissenbacher. The alvis format for linguistically annotated documents. *arXiv preprint cs/0609136*, 2006.

[58] Günter Neumann and Bogdan Sacaleanu. Experiments on robust nl question interpretation and multi-layered document annotation for a cross–language question/answering system. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 411–422. Springer, 2004.

[59] LIONEL NICOLAS, AIVARS GLAZNIEKS, EGON STEMLE, and ANDREA ABEL. A generic data workflow for building annotated text corpora, 2015.

[60] Philip V Ogren, Philipp G Wetzler, and Steven Bethard. Cleartk: A uima toolkit for statistical natural language processing. *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP*, 32, 2008.

[61] Petya Osenova and Sia Kolkovska. Combining the named-entity recognition task and np chunking strategy for robust pre-processing. In *Proceedings of the Workshop on Treebanks and Linguistic Theories, September*, pages 20–21, 2002.

[62] Subarno Pal and Soumadip Ghosh. Sentiment analysis using averaged histogram. *International Journal of Computer Applications*, 162(12), 2017.

[63] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[64] Maria Teresa Pazienza, Marco Pennacchiotti, and Fabio Massimo Zanzotto. Terminology extraction: an analysis of linguistic and statistical approaches. In *Knowledge mining*, pages 255–279. Springer, 2005.

[65] Georgios Petasis. Ellogon and the challenge of threads. In *Proceedings of the 17'th Annual Tcl/Tk Conference*, page 287. Lulu. com, 2010.

[66] Georgios Petasis, Vangelis Karkaletsis, Georgios Paliouras, Ion Androutsopoulos, and Constantine D Spyropoulos. Ellogon: A new text engineering platform. *arXiv preprint cs/0205017*, 2002.

[67] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, pages 91–100. ACM, 2008.

[68] Raymond K Pon, Alfonso F Cardenas, David Buttler, and Terence Critchlow. Tracking multiple topics for finding interesting articles. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 560–569. ACM, 2007.

[69] Piotr Przybyła, Austin J Brockmeier, Georgios Kontonatsios, Marie-Annick Le Pogam, John McNaught, Erik von Elm, Kay Nolan, and Sophia Ananiadou. Prioritising references for systematic reviews with robotanalyst: A user study. *Research synthesis methods*, 9(3):470–488, 2018.

[70] Dietrich Rebholz-Schuhmann, Harald Kirsch, Goran Nenadic, D Rebholz-Schuhmann, H Kirsch, and G Nenadic. Iexml: towards an annotation framework for biomedical semantic types enabling interoperability of text processing modules. *SIG BioLink, ISMB*, 2006.

[71] Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. Nlpport: A pipeline for portuguese nlp (short paper). In *OASIcs-OpenAccess Series in Informatics*, volume 62. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[72] Carlos Rodrıguez-Penagos, David Garcıa Narbona, Guillem Massó Sanabre, Jens Grivolla, and Joan Codina Filbá. Sentiment analysis and visualization using uima and solr. *Unstructured Information Management Architecture (UIMA)*, page 42, 2013.

[73] Sara Rosenthal, Noura Farra, and Preslav Nakov. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518, 2017.

[74] Hassan Saif, Yulan He, and Harith Alani. Semantic sentiment analysis of twitter. In *International semantic web conference*, pages 508–524. Springer, 2012.

[75] Diana Santos. Punctuation and multilinguality: Some reflections from a language engineering perspective. *Working Papers in Applied Linguistics*, 4(98):138–160, 1998.

[76] Abeed Sarker and Graciela Gonzalez. Hlp @ upenn at semeval-2017 task 4a: A simple, self-optimizing text classification system combining dense and sparse vectors. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 640–643, 2017.

[77] G Savova, Karin Kipper-Schuler, J Buntrock, and C Chute. Uima-based clinical information extraction system. *Towards enhanced interoperability for large HLT systems: UIMA for NLP*, 39, 2008.

[78] Guergana K Savova, James J Masanz, Philip V Ogren, Jiaping Zheng, Sunghwan Sohn, Karin C Kipper-Schuler, and Christopher G Chute. Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 17(5):507–513, 2010.

[79] Ulrich Schäfer. Middleware for creating and combining multi-dimensional nlp markup. In *Proceedings of the 5th Workshop on NLP and XML: Multi-Dimensional Markup in Natural Language Processing*, pages 81–84. Association for Computational Linguistics, 2006.

[80] Fadi Abu Sheika and Diana Inkpen. Learning to classify documents according to formal and informal style. *Linguistic Issues in Language Technology*, 8(1):1–29, 2012.

[81] Sunghwan Sohn and Guergana K Savova. Mayo clinic smoking status classification system: extensions and improvements. In *AMIA Annual Symposium Proceedings*, volume 2009, page 619. American Medical Informatics Association, 2009.

[82] UMICH. UMICH SI650 - Sentiment Classification. `https://www.kaggle.com/c/si650winter11`, 2011. Accessed: 2018-12-20.

[83] Joan-Josep Vallbé, M Antònia Martí, Blaz Fortuna, Aleks Jakulin, Dunja Mladenic, and Pompeu Casanovas. Stemming and lemmatisation: improving knowledge management through language processing techniques. *Trends in Legal Knowledge, the Semantic Web and the Regulation of Electronic Social Systems*, 2007.

[84] Fabio Valsecchi, Matteo Abrate, Clara Bacciu, Silvia Piccini, and Andrea Marchetti. Text encoder and annotator: an all-in-one editor for transcribing and annotating manuscripts with rdf. In *International Semantic Web Conference*, pages 399–407. Springer, 2016.

[85] Johanna Völker, Sergi Fernandez Langa, and York Sure. Supporting the construction of spanish legal ontologies with text2onto. In *Computable Models of the Law*, pages 105–112. Springer, 2008.

[86] Benjamin Waldron, Ann Copestake, Ulrich Schäfer, and Bernd Kiefer. Preprocessing and tokenisation standards in delph-in tools. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pages 2263–2268, 2006.

[87] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

[88] Stephen T Wu, Vinod C Kaggal, Dmitriy Dligach, James J Masanz, Pei Chen, Lee Becker, Wendy W Chapman, Guergana K Savova, Hongfang Liu, and Christopher G Chute. A common type system for clinical natural language processing. *Journal of biomedical semantics*, 4(1):1, 2013.

[89] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.

[90] Yang Yu and Xiao Wang. World cup 2014 in the twitter world: A big data analysis of sentiments in us sports fans' tweets. *Computers in Human Behavior*, 48:392–400, 2015.

[91] Wajdi Zaghouani. Renar: A rule-based arabic named entity recognition system. *ACM Transactions on Asian Language Information Processing (TALIP)*, 11(1):2, 2012.

[92] Zhi Zhong and Hwee Tou Ng. It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of the ACL 2010 system demonstrations*, pages 78–83. Association for Computational Linguistics, 2010.

A NLP Tasks and Data Types

The following table presents the Natural Language Processing (NLP) tasks that relate to the data and requirements of Typhon use case partners. Each task is accompanied by a description of its function and the prerequisite NLP tasks that need to run before it. For example, in order to split the sentences of a text, we first need to read the contents first and thus the prerequisite component for a Sentence Segmentation is a Document Reader: T1->T3. The last column shows the data type of the result of the execution of each NLP task which will be subsequently stored and indexed in ElasticSearch.

NLP Task	Id	Description	Pre-requisite NLP Tasks	Generated ElasticSearch Object
Document Reader	T1	Reads a document from a sourceFile location and creates a Document data type		Document + begin: Int + end: Int + language: String + encoding: String + sourceFile: URI
Paragraph Segmentation	T2	Segments input document into constituent paragraphs	T1 → T2	Paragraph + begin: Int + end: Int
Sentence Segmentation	T3	Segments input document or paragraph into constituent sentences	T1 → T3	Sentence + begin: Int + end: Int + sentenceValue: String
Tokenisation	T4	Segments input document/paragraph/sentence into constituent words	T1 → T3 → T4	Token + begin: Int + end: Int + tokenValue: String
Phrase Extractor	T5	Segments input document/paragraph/sentence into constituent phrases	T1 → T3 → T4 → T5	Phrase + begin: Int + end: Int + phraseValue: List<Token>
NGram Extractor	T6	Segments input document/paragraph/sentence into constituent n-grams (i.e. list of n consecutive words)	T1 → T3 → T4 → T6	NGram + begin: Int + end: Int + ngramValue: List<Token>
PoS Tagging	T7	Identifies Part-of-Speech tags of tokens	T1 → T3 → T4 → T7	POS + begin: Int + end: Int + posValue: String
Lemmatisation	T8	Converts tokens into lemmas, e.g. computing → compute	T1 → T3 → T4 → T8	Lemma + begin: Int + end: Int + lemmaValue: String
Stemming	T9	Converts tokens into stems, e.g. computing → comput	T1 → T3 → T4 → T9	Stem + begin: Int + end: Int + stemValue: String

NLP Task	Id	Description	Pre-requisite NLP Tasks	Generated ElasticSearch Object
Dependency Parsing	T10	Identifies syntactic relationships between "source" words and "target" words which modify those "source" words. The label of the syntactic relationship describes the exact nature of the dependency.	T1 → T3 → T4 → T7 → T10	Dependency + begin: Int + end: Int + source: Token + target: Token + label: String
Chunking	T11	Identifies syntactic groups, i.e. chunks, consisting of tokens with their corresponding PoS tags.	T1 → T3 → T4 → T7 → T11	Chunk + begin: Int + end: Int + constituentTokens: List<Token> + constituentPOS: List<POS> + label: String
Sentiment Analysis	T12	Assigns a polarity label (e.g. positive/neutral/negative) or a polarity score (Float) to a document	T1 → T3 → T4 → T8 → T6 → T12	Polarity + begin: Int + end: Int + polarityLabel: String + polarityScore: Float
Text Classification	T13	Assigns a category (e.g. politics/news/sports) or a category score (Float) to a document	T1 → T3 → T4 → T8 → T6 → T13	Cateory + begin: Int + end: Int + categoryLabel: String + categoryScore: Float
Topic Modelling	T14	Assigns a topic(s) to an input document. A topic is described by a list of descriptive keyword. A descriptive keyword is assigned a coefficient score denoting the correlation between the word and the underlying topic	T1 → T3 → T4 → T14	Topic + begin: Int + end: Int + descriptiveWords: List<String> + wordCoefficients: List<Float>
Term Extraction	T15	Extracts terms (single-word or multi-word) from an input document. A term is also assigned a weight denoting the importance of a term within a document	T1 → T3 → T4 → T7 → T15	Term + begin: Int + end: Int + constituentTokens: List<Token> + weight: Float
Named Entity Recognition	T16	Extracts named entities (single-word or multi-word) from an input document. A named entity is assigned a semantic role (e.g. product/organisation/location/person name)	T1 → T3 → T4 → T7 → T11 → T16	NamedEntity + begin: Int + end: Int + constituentTokens: List<Token> + label: String
Relation Extraction	T17	Identifies semantic links/relations between a source and a target NamedEntity.	T1 → T3 → T4 → T7 → T11 → T10 → T16 → T17	Relation + begin: Int + end: Int + sourceNE: NamedEntity + targetNE: NamedEntity + label: String

NLP Task	Id	Description	Pre-requisite NLP Tasks	Generated ElasticSearch Object
Event Extraction	T18	Extracts events from an input document. An event (i.e. textual fact) is "triggered" by a word and it links two or more NamedEntities (arguments of event)	T1 → T3 → T4 → T7 → T11 → T10 → T16 → T18	Event + begin: Int + end: Int + triger: token + arguments: List<NamedEntity> + theme: String
Coreference Resolution	T19	Extracts coreference chains from an input document. A coreference chain consists of an antecedent, i.e. the main entity referred in text, and an anaphora, e.g. a pronoun referring to the antecedent	T1 → T3 → T4 → T7 → T11 → T10 → T16 → T19	CoreferenceChain + begin: Int + end: Int + antecedent: List<Token> + anaphora: List<Token>

Table 18: NLP tasks and associated data types