D3.5 - EW-Shopp Components as a Service: Final Release

| Deliverable n: | 3.5 |
| Delivered date: | 31 December 2019 |
| Status: | Final |
| Version: | 1.0 |
| Authors: | Lorenzo Sutton (ENG), Nikolay Nikolov (SINTEF), Michele Ciavotta (UNIMIB), Aljaž Košmerlj (JSI) |
| Contributors: | UNIMIB, ENG, SINTEF, JSI |
| Reviewers: | David Čreslovenik (CE) |
| Distribution: | Public |
| Type: | OTHER |
## History of Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>9.10.2019</td>
<td>Definition of Table of Contents</td>
<td>Lorenzo Sutton (ENG)</td>
</tr>
<tr>
<td>0.11</td>
<td>10.10.2019</td>
<td>Updated ToC</td>
<td>Lorenzo Sutton (ENG)</td>
</tr>
<tr>
<td>0.12</td>
<td>23.10.2019</td>
<td>Finalising ToC at the general meeting</td>
<td>Lorenzo Sutton (ENG), UNIMIB, JSI, SINTEF</td>
</tr>
<tr>
<td>0.13</td>
<td>6.12.2019</td>
<td>Updated version with most partner contributions</td>
<td>ENG, SINTEF, UNIMIB, JSI</td>
</tr>
<tr>
<td>0.14</td>
<td>13.12.2019</td>
<td>Updates to section 2 and 4 as well as 1 and 5. Ready for internal review</td>
<td>ENG, SINTEF</td>
</tr>
<tr>
<td>0.15</td>
<td>18.12.2019</td>
<td>Internal reviews</td>
<td>CE, UNIMIB</td>
</tr>
<tr>
<td>0.16</td>
<td>19.12.2019</td>
<td>Updated version addressing internal review comments</td>
<td>ENG</td>
</tr>
<tr>
<td>0.17</td>
<td>23.12.2019</td>
<td>Appendix and Chapter 5 added</td>
<td>UNIMIB</td>
</tr>
<tr>
<td>1.0</td>
<td>28.12.2018</td>
<td>Final check and minor editing by coordinator</td>
<td>Matteo Palmonari (UNIMIB)</td>
</tr>
</tbody>
</table>
**Executive Summary**

This is the accompanying report for deliverable D3.5 *EW-Shopp components as a service: final release*. It provides information about all of the main EW-Shopp components, which have been released as part of an integrated yet modular toolkit. Following the Dataflow developed in EW-Shopp methodology, and its related logical structure, the document presents the toolkits and related components and services in the three key phases of the dataflow: 1) Data preparation and enrichment; 2) Data Analytics and 3) Data visualisation and navigation. As of M36 (end of the project period) the EW-Shopp toolkit is made of software available as open source.

The document, after an introductory chapter, is divided into four chapters. In Chapter 2 services aimed at complete data preparation and enrichment are described, namely Grafterizer, ASIA and ABSTAT: the first two tools are used to improve the quality of the data to be analysed, while the third one supports quality assessment of RDF graph-based data, which are often used for data enrichment. Chapter 3 provides an overview of the Data Analytics tools, which consist in an Event and Weather Analytics toolkit built on top of QMiner (subject to only minor business-case specific updates with respect to previous deliverables), and Keyword Clustering and Media Attention tools, two new analytic features added recently to the EW-Shopp toolkit. In Chapter 4, the Knowage open source suite is presented as the data visualisation and navigation service. Finally, in Chapter 5, we present three more EW-Shopp novel technical contributions, that is, algorithms and services that are not components of the EW-Shopp toolkit but support data curation and innovation in the data value chains addressed in the project.

The first three chapters present the tools related to each of the above stages with factsheets highlighting the main features, information about their architectures (when relevant of if updated with respect to previous deliverables) and examples of usage (when not provided in previous deliverables). Since a characteristic of the toolkit is that it is made in a modular way, meaning that while all the tools have been tested together and in organic workflows within EW-Shopp business cases, the use of all tools together is not enforced, allowing to potentially picking the best combination of components depending on users’ use cases. This document therefore focuses on the tools as individual components and on their functionalities, leaving more technical explanations about how to orchestrate these services into organic work flows and how to deploy these work flows on cloud platforms for large-scale data processing to the document that discuss Deliverable 2.4, which provides the final release of the toolkit (formerly known as EW-Shopp platform).

The work described in this report, and the components themselves, fulfil the requirements and follows the technical specifications described in deliverables from work package 1. The services are part of the components in the overall toolkit architecture developed within work package 2 and they were demonstrated in the business cases carried out in work package 4.
Tale of contents

History of Changes ........................................................................................................................................... 2
Executive Summary ............................................................................................................................................. 3
List of figures ..................................................................................................................................................... 7
List of Tables .................................................................................................................................................... 9
Acronyms ......................................................................................................................................................... 10
Chapter 1  Introduction .................................................................................................................................. 12
  1.1  Objectives and Scope .......................................................................................................................... 12
  1.2  Relationship to Other Deliverables ........................................................................................................ 14
Chapter 2  Data Preparation and Enrichment ............................................................................................... 15
  2.1  Data Preparation and Enrichment: Workflow ....................................................................................... 15
  2.2  EW-Shopp Data Preparation and Enrichment Services ........................................................................ 16
    2.2.1  Grafterizer ...................................................................................................................................... 17
    2.2.2  ASIA ............................................................................................................................................... 21
    2.2.3  ABSTAT ....................................................................................................................................... 28
  2.3  Key Updates Since the Previous Release ............................................................................................... 30
    2.3.1  Updates to ASIA ............................................................................................................................ 30
Chapter 3  Data Analytics ............................................................................................................................... 33
  3.1  Event and Weather Analytics Toolset ................................................................................................... 33
  3.2  Keyword Clustering Tool ....................................................................................................................... 34
    3.2.1  Factsheet ...................................................................................................................................... 35
    3.2.2  Approach ...................................................................................................................................... 36
    3.2.3  Implementation ............................................................................................................................... 36
    3.2.4  Workflow ...................................................................................................................................... 37
    3.2.5  Usage ............................................................................................................................................ 38
    3.2.6  JOT Case Optimization .................................................................................................................. 41
  3.3  Media Attention Tool .............................................................................................................................. 42
    3.3.1  Factsheet ...................................................................................................................................... 42
    3.3.2  Media Attention Features ............................................................................................................. 43
    3.3.3  Usage ............................................................................................................................................ 45
    3.3.4  Relation to the Custom Events Model ........................................................................................... 48
  3.4  Key Updates Since the Previous Release ............................................................................................... 48
A.4.3. Discussions and Conclusion ........................................................................... 98
A.5. Conclusion ......................................................................................................... 98
A.5.1. Comparison .................................................................................................... 100
A.5.2. Discussion on the Pipeline ............................................................................ 102
List of figures

Figure 1 General Dataflow methodology and its steps — Source: EW-Shopp D2.3 .......................................................... 13
Figure 2: Data transformation and enrichment from the data quality viewpoint ............................................................... 15
Figure 3. Grafterizer high-level architecture .................................................................................................................. 19
Figure 4. Scalable back-end for Grafterizer .................................................................................................................... 20
Figure 5: ASIA integrated in Grafterizer .......................................................................................................................... 25
Figure 6: Schema-level linking architectural view ........................................................................................................ 25
Figure 7. The interlinking service architecture ............................................................................................................... 26
Figure 8: ASIA’s keyword2category reconciliation widget .............................................................................................. 31
Figure 9: Example of keywords matched with their closest category ............................................................................. 32
Figure 10 Main Knowage website (left): www.knowage-suite.com — EW-Shopp Knowage 7.0 instance login page
(right) ........................................................................................................................................................................... 50
Figure 11 Definition of a Data Source in Knowage — in this case a MySQL database for Business Case 3 (Measureuse) ......... 53
Figure 12 Accessing Data sets from the main Knowage 7.0 menu ................................................................................ 54
Figure 13 CSV Data Set for Business Case 3 (Measureuse) ............................................................................................... 54
Figure 14 SQL query Data Set for business case 3. Notice in the SQL snippet shown, the dynamic parameters set with
the syntax e.g. $P{storID}$ and $P{day_week_1}$ ............................................................................................................ 55
Figure 15 A REST data set in Knowage, in this case the ArangoDB connector for JOT Business Case 4. Also in this
example a dynamic parameter is used .......................................................................................................................... 56
Figure 16 Data set preview for one of the BC3 data sets. If the query contains parameters the user is able to insert
them before previewing or use the default values. ........................................................................................................ 56
Figure 17 Defining role metadata (attribute or measure) for one of the BC3 data sets .................................................... 57
Figure 18 Accessing the LOV menu. Configuration of a LOV analytical driver (above) and related selection of the
driver within a document set-up (below) ...................................................................................................................... 58
Figure 19 Empty Cockpit creation ‘canvas’ and selection of available widgets to be added........................................... 59
Figure 20 Adding two data sets from EW-Shopp Business Case 3. Notice that for the query data set the parameters
are exposed .................................................................................................................................................................... 59
Figure 21 Visually creating parameters (i.e. columns) associations between data sets in a cockpit .................................. 60
Figure 22 Setting up a chart for the Business Case 3 weekly store comparison cockpit ................................................ 60
Figure 23 Quickly setting visual features of the chart ...................................................................................................... 61
Figure 24 Widgets can be dragged, edited, and copied around the cockpit canvas in a WYSIWYG way ..................... 61
Figure 25 Editing of the HTML widget. Part of the HTML is templated and injected data from the data set ............... 62
Figure 26 Parametric text widget. The actual text generated in the cockpit is extracted dynamically from the
dataset ......................................................................................................................................................................... 62
Figure 27 Two examples of parameter selection: left checkbox style (used in BC3). Right: searching within the BC11
product catalogue. Once selected the parameters will be used with the connected datasets ................................ 63
Figure 28 Query parameters for LOVs used in the cockpit can be saved for quick retrieval ......................................... 63
Figure 29 The selected parameter is used in most widgets using the same data set highlighted by the yellow boxes: in
this example the title text, the chart and the HTML widget all rely on the parametrized selection of the store
location ...................................................................................................................................................................... 64
Figure 30 Advanced document editing for the cockpit including (below) Analytical driver set-up ................................ 64
Figure 31 Knowage user management menu (above) and interface (below) ................................................................. 65
Figure 32 Setting up a typical EW-Shopp business case user role in the administration interface ............................. 66
Figure 33 Server configuration interface, including incremental search of parameters .............................................. 66

EW-Shopp GA number: 732590 H2020-ICT-2016-2017/H2020-ICT-2016-1
Figure 34: Retrieving a Knowage data set via REST API in Postman (above) and with Python requests (below) .... 68
Figure 35: Example of the Measurence store analysis embedded in an eternal HTML page .................. 69
Figure 36: DeepMatcher data flow — from SIGMOD18 paper ......................................................... 72
Figure 37: DeepMatcher configuration space — from SIGMOD18 paper ............................................. 73
Figure 38: Example of training data fed to DeepMatcher where the attributes Title and Brand are used as attributes ................................................................. 76
Figure 38: Excerpt showing a subset of vocabularies used .................................................................. 82
Figure 40: D2R server homepage of GfK Product Catalog ............................................................... 83
Figure 41: Catalog’s HTML interface ..................................................................................................... 83
Figure 42: SPARQL interface ................................................................................................................ 84
Figure 43: SPARQL query example ....................................................................................................... 84
Figure 44: Product data sheet .............................................................................................................. 85
Figure 45: Search via product category ................................................................................................ 85
Figure 46: MantisTable interface overview: Visualization Mode (1. left side bar, 2. console), Info Mode (3. right side bar), Edit Mode (4. edit form) ............................................................. 88
Figure 47: Example of enriching a dataset with temperature data ........................................................ 90
Figure 48: EW-Shopp scenario (based on tutorial) .............................................................................. 91
Figure 49: Join feature in Trifacta .......................................................................................................... 92
Figure 50: Screenshot of Trifacta UI from a different project ............................................................. 93
Figure 51: Screenshot of Trifacta Sample feature ................................................................................ 93
Figure 52: Screenshot of the KNIME pipeline ..................................................................................... 95
Figure 53: Screenshot of KNIME String Matcher setting ................................................................. 96
Figure 54: Screenshot of KNIME string matcher results ................................................................. 96
Figure 55: Screenshot of Talend pipeline ........................................................................................... 98
Figure 56: First analysis on enrichment values ................................................................................... 99
Figure 57: Deeper analysis on city enrichment .................................................................................. 100
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>List of acronyms used throughout the document</td>
<td>10</td>
</tr>
<tr>
<td>Table 2</td>
<td>Short references for project partners</td>
<td>11</td>
</tr>
<tr>
<td>Table 3</td>
<td>Grafterizer – Fact sheet</td>
<td>17</td>
</tr>
<tr>
<td>Table 4</td>
<td>Grafterizer – Summary of features</td>
<td>20</td>
</tr>
<tr>
<td>Table 5</td>
<td>ASIA – Fact sheet</td>
<td>22</td>
</tr>
<tr>
<td>Table 6</td>
<td>ASIA - Summary of features</td>
<td>26</td>
</tr>
<tr>
<td>Table 7</td>
<td>ABSTAT – Fact sheet</td>
<td>29</td>
</tr>
<tr>
<td>Table 8</td>
<td>Event and Weather Analytics Toolset – Fact sheet</td>
<td>33</td>
</tr>
<tr>
<td>Table 9</td>
<td>Keyword Clustering Tool – Fact sheet</td>
<td>35</td>
</tr>
<tr>
<td>Table 10</td>
<td>Media Attention tool - factsheet</td>
<td>42</td>
</tr>
<tr>
<td>Table 11</td>
<td>Knowage – Fact sheet</td>
<td>51</td>
</tr>
<tr>
<td>Table 12</td>
<td>Main activities for data enrichment</td>
<td>101</td>
</tr>
<tr>
<td>Table 13</td>
<td>Other characteristics</td>
<td>101</td>
</tr>
<tr>
<td>Table 14</td>
<td>Structure</td>
<td>102</td>
</tr>
<tr>
<td>Table 15</td>
<td>Other information</td>
<td>102</td>
</tr>
</tbody>
</table>
**Acronyms**

Table 1: list of acronyms used throughout the document

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BC</td>
<td>Business Case</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>VDP</td>
<td>Visual Data Profiling</td>
</tr>
<tr>
<td>JAR</td>
<td>Java ARchive</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>GLCI-RDF</td>
<td>Google GeoTargets in RDF</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>HDT</td>
<td>Header Dictionary Triples</td>
</tr>
<tr>
<td>LOV</td>
<td>List Of Values</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
</tbody>
</table>

Short references may be used to refer to project beneficiaries, also frequently referred to as partners. References are listed in Table 2.
Table 2. Short references for project partners

<table>
<thead>
<tr>
<th>No.</th>
<th>Beneficiary (partner) name as in [GA]</th>
<th>Short reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA</td>
<td>UNIMIB</td>
</tr>
<tr>
<td>2</td>
<td>CENEJE DRUZBA ZA TRGOVINO IN POSLOVNO SVETOVANJE DOO</td>
<td>CE</td>
</tr>
<tr>
<td>3</td>
<td>BROWSETEL (UK) LIMITED</td>
<td>BT</td>
</tr>
<tr>
<td>4</td>
<td>GFK EURISKO SRL.</td>
<td>GFK</td>
</tr>
<tr>
<td>5</td>
<td>BIG BANG, TRGOVINA IN STORITVE, DOO</td>
<td>BB</td>
</tr>
<tr>
<td>6</td>
<td>MEASURENCE LIMITED</td>
<td>ME</td>
</tr>
<tr>
<td>7</td>
<td>JOT INTERNET MEDIA ESPAÑA SL</td>
<td>JOT</td>
</tr>
<tr>
<td>8</td>
<td>ENGINEERING – INGEGNERIA INFORMATICA SPA</td>
<td>ENG</td>
</tr>
<tr>
<td>9</td>
<td>STIFTELEN SINTEF</td>
<td>SINTEF</td>
</tr>
<tr>
<td>10</td>
<td>INSTITUT JOZEF STEFAN</td>
<td>JSI</td>
</tr>
<tr>
<td>11</td>
<td>GfK Italy</td>
<td>GFK¹</td>
</tr>
<tr>
<td>12</td>
<td>SINTEF AS</td>
<td>SINTEF²</td>
</tr>
</tbody>
</table>

¹ As a result of change in the GFK EURISKO SRL, the partner GFK Italy has implemented the activities assigned to GFK EURISKO SRL since 1/7/2018. This has been agreed between the EW-Shopp consortium and the EC, leading to an amendment of the Grant Agreement. Since the actions of the two entities are contiguous and not overlapping we use the same acronym GFK for both the entities.

² SINTEF AS has partially taken over STIFTELEN SINTEF activities, and in particular, it took over the activities since 01/01/2018, as agreed between the EW-Shopp consortium and the EC leading to an amendment of the Grant Agreement. Since the actions of the two entities are contiguous and not overlapping we use the same acronym SINTEF for both the entities.
Chapter 1  Introduction

This is the accompanying report for deliverable EW-Shopp deliverable D3.5 *EW-Shopp components as a service: final release*. The deliverable is constituted by the final release of the various EW-Shopp components (and is therefore classified as OTHER); nonetheless, with this report we aim to provide a detailed insight in such components, in particular of such final version. The components have been organised in the *EW-Shopp toolkit* components (formerly ‘platform’), as of month 36 of the project. Objective of the toolkit components is to support users in generating value out of their data by enriching them via external data sources, visualizing and exploring them, and finally producing insights and operational models via machine learning. Alongside reporting the descriptions and main features of the toolkit, the report also presents some specific dataflow (and usage scenarios). Finally, at the end of this document are presented some tools and algorithms developed during the project to meet the needs of the business cases and that, despite being main results of the project and assets valuable and reusable in other contexts, have not found their place within the main components of the toolkit. They are therefore presented here for the first time.

1.1 Objectives and Scope

To facilitate both the conceptualisation and the practical operation of complex data pipelines in EW-Shop, we decided to rationalise the process from a logical / operational point of view (but also from a hands-on one) through its application to the EW-Shopp business cases. To this end we consolidated a shared EW-Shopp Dataflow methodology which was already presented in detail in D2.3, in the context of assessment, where the rationalisation and organization of the overall flows for each Business Case were necessary on the one hand to obtain smooth data flows and on the other hand to have a consistent framework for carrying out and evaluating these EW-Shopp-based business cases. Below we report the general figure for the dataflow (Figure 1).

In presenting the toolkit services we still generally follow the Dataflow methodology in assuming that a typical business/operational scenario supported by EW-Shopp will carry out all of the steps, albeit with different approaches and requirements as explained also in other deliverables, and in particular D2.3. In brief the dataflow steps are the following: 1) **Ingestion** where the data are acquired and possibly aggregated; 2) **Enrichment** where the various ‘raw’ data coming from different sources and in different formats are filtered, reconciled, transformed, aggregated, normalized, etc. in order to make them more usable within the following steps; 3) **Analytics** where several algorithms and tools are used to analyse the data obtained through the previous steps and generate models and analyses; 4) **Visualisation** where data analysed in the previous step is finally visualised, navigated and explored and actual data analytics results are presented to final users.
Aim of the EW-Shopp services is to support such dataflow, in particular where business data are being used in conjunction with weather and/or events data. Therefore, in this document we report the various services following the organisation of the Dataflow steps.

Chapter 2 covers steps 1) and 2) of the Dataflow describing the Data Transformation, Linking and Quality services.

Chapter 3 describes Data Analytics services in EW-Shopp, in particular the Keyword Clustering Tool supporting clustering tasks, and analysis of media events, in particular their use in the marketing domain, and the relation to custom event models.

Chapter 4 describes Visualisation Services, in particular trying to provide hands-on information on how to visualise time-based data series which incorporate business metrics together with weather data, a scenario common to most EW-Shopp business cases and any other similar one even beyond EW-Shopp itself.

Chapter 5 presents other supporting tools and miscellaneous work not attributable in particular to any one of the main services comprising the toolkit but relevant to cover unforeseen needs emerged during the project.
Moreover, each chapter also emphasises the services’ updates compared to the previous releases (respectively at M18 – D3.2 and M24 – D3.3).

Finally, an appendix to this document presents a comparative assessment of the enrichment component performed to accompany and complement the one presented in D3.4.

### 1.2 Relationship to Other Deliverables

As the final release of the EW-Shopp toolkit D3.5 builds upon all previous WP3 deliverables and in particular D3.1 where the initial specifications and designs were defined, D3.2 and D3.3 which were the initial releases of the services. The components’ evaluation carried out in D3.4 as well as the platform (toolkit) assessments carried out in D2.3 were used to capture the missing requirements of the service and steer the development towards this release. As anticipated above, D2.3 also served to consolidate the Dataflow methodology among partners and for subsequent work. The services are, clearly, also tightly coupled with the final Toolkit release delivered as D2.4 wherein the focus is on component integration and usage rather than on the features implemented by each of them, which is the ultimate goal of this deliverable. Deliverables 3.5 and 2.4 complement each other in that D3.5 aims to provide the final view of all toolkit components as well as other tools and services developed in EW-Shopp (see Chapter 5 for specifics); D2.4, on the other hand, provides the actual release and therefore also has a more deployment and ‘hands-on’ technical information.
Chapter 2 Data Preparation and Enrichment

This chapter discusses the data preparation and enrichment services of the EW-Shopp toolkit, which provides functionalities for scalable clean-up, transformation and enrichment of data. Unlike other deliverables where services have already been presented in detail, to avoid duplication in this chapter we will provide a high-level presentation of services and their use (workflow) highlighting where necessary the differences and novelties compared to what is written in D3.3. In addition, summary tables will be presented to make the presentation more immediate.

2.1 Data Preparation and Enrichment: Workflow

Figure 2: Data transformation and enrichment from the data quality viewpoint

In EW-Shopp, the data preparation and enrichment services are primarily involved in the transformation of the tabular data set to be passed as input to some data analytics tools. Figure 2 illustrates the pipeline defined in the project that, starting from a raw (csv alike) data set, cleans, annotates, enriches and (optionally) stores it in RDF format; this process is eventually executed on a large scale. Notice that next to each block representing a workflow activity the main services involved are reported. Below, we briefly analyse the workflow activities.

- **Data Cleaning and Transformation:** during this phase the initial data set is cleaned and transformed. Typical transformations are the dropout of wrong or useless (from the point of
view of the analytics to be built upon) data and the addition of new columns with derived data if needed. In this phase, the user intrinsically acts to create a data set of higher quality than the one (s)he started with. A Grafterizer component, referred to as Visual Data Profiling, guides the user along this path through a dashboard that simplifies the identification of quality issues (quality assessment). Notice that, whereas during EW-Shopp several new functionalities have been added to Grafterizer, this feature was not developed in the EW-Shopp project frame.

- **Schema-level linking:** in this phase an RDF mapping is created that associates the columns of the tabular data set resulting from the previous activity to classes and datatypes of specific vocabularies. ABSTAT and ASIA work together to suggest to the user the best match. Also, in this case the functionality is pre-existing the project EW-Shopp.

- **Instance-level linking and data enrichment:** in this phase, the work data set is enriched by reconciling a column with an external system of identifiers (what we refer to as instance-level linking) in order to extend the data with information available in some reference knowledge base that uses those identifiers (enrichment). ASIA, through an intuitive graphical interface embedded in Grafterizer, can support users in performing this task. An indicator that identifies the accuracy of the matching results and allows the user to discard matchings with insufficient accuracy is also available. The ASIA’s functionalities exploited in this phase have been developed from scratch within the project.

- **Quality assessment:** The transformation and enrichment process may conclude with the publication of the data set in a semantic format; in particular, Grafterizer, applying user-defined transformations and RDF mappings, is able to generate a file in N-Triples format\(^3\) or save the results in a json-based graph format in an ArangoDB database\(^4\). Once data are published it is possible to assess the final quality of data through a profiling task. In case data are produced in RDF, since ABSTAT supports natively RDF and it can be used to profile the data set to identify possible quality issues.

- **Large scale processing:** Grafterizer allows the user to save the transformation and enrichment pipeline described in the previous steps in different formats; actually, this it can also generate an executable application in jar format that can be reused to replay the user-defined transformations on a separate dataset (however with the same structure). As part of the EW-Shopp project a new component (Scalable data preparation and enrichment back-end) has been developed that uses the automatically generated executable code along with the best practices of Big Data processing to transform and enrich larger datasets (tests with datasets up to one hundred gigabytes have been performed).

## 2.2 EW-Shopp Data Preparation and Enrichment Services

---

\(^3\) [https://www.w3.org/TR/n-triples/](https://www.w3.org/TR/n-triples/)

\(^4\) [https://www.arangodb.com/](https://www.arangodb.com/)
In this section, we describe data preparation and enrichment services in a concise yet analytical form, delivering a description of their objectives, architecture and functionalities (both pre-existing and developed within the project). More details, including explanatory examples and API descriptions, can be found in deliverables D3.3 and D2.4, in published papers\(^5\)\(^6\), as while a video showing data transformation and enrichment in action using Grafterizer and ASIA is available on YouTube\(^7\).

### 2.2.1 Grafterizer

Grafterizer\(^8\) is a web-based framework that, together with ASIA, provides the user with the tools to undertake data transformation, data cleaning and semantic enrichment operations for tabular data cleaning, transformation, and RDF mapping. Grafterizer is a component of a larger software ecosystem for Open Data management named DataGraft\(^9\). DataGraft also provides some pre-existing data hosting and publishing capabilities, user account management, and dataset and database management.

Grafterizer is used for data cleaning and, possibly data conversion into a semantic format. The data transformation includes steps for both cleaning data and mapping to different export formats. Transformation steps include row-based operations (add, drop, filter, duplicate detection, etc.), column-based (add, drop, rename, merge, etc.) and operations on the entire dataset (sort, aggregate, etc.) and results are displayed after each step is applied. Grafterizer provides capabilities to semantically map the tabular data resulting from data preparation and cleaning to existing vocabularies (e.g., schema.org) or to user-defined ones. The result can be exported as RDF triples or as JSON collections of edges and values that can be imported into the multi-model database ArangoDB. Grafterizer is based on the Clojure library Grafter\(^10\), which enables Grafterizer code to be ran on the Java Virtual Machine. This makes it possible to generate and executable Java ARchive (JAR) files for each transformation that can be executed offline, for example, in order to scale transformation execution and treat large data sets.

ArangoDB databases, collections, and graphs are handled in DataGraft using registration of ArangoDB assets, which are managed in the same way as the other assets on the platform (e.g., SPARQL\(^11\) endpoints for RDF data). ArangoDB assets on DataGraft allow users to read, update and delete collections from a specific database. The ArangoDB integration also allows to directly upload transformed data (using Grafterizer) to collections in the database.

#### 2.2.1.1 Factsheet

Table 3: Grafterizer – Fact sheet

\(^5\) https://link.springer.com/chapter/10.1007/978-3-030-30796-7_22  
\(^6\) http://ceur-ws.org/Vol-2456/paper54.pdf  
\(^7\) https://www.youtube.com/playlist?list=PLy7SznldqmezwdL4QcxQYy2Fz1HV0wMS  
\(^8\) https://github.com/datagraft/grafterizer-2.0/  
\(^9\) https://datagraft.io/  
\(^10\) http://grafter.org  
\(^11\) https://www.w3.org/TR/sparql11-query/
Grafterizer is a graphical data preparation and transformation tool that allows for storage and retrieval of data transformation scripts and CSV files, reconciliation of tabular data with external knowledge bases (through ASIA integration), data extension (through ASIA integration) and for preparation and processing of large volumes of data by providing a scalable transformation execution architecture.

In EW-Shopp Grafterizer has been used for various data preparation and transformation tasks related to business cases and preparation/serialization of weather data that can be used for enrichment. Furthermore, the tool was used to reconcile against external knowledge bases such as GeoNames and DBpedia and for enrichment of business case input datasets with weather and event data.

**Usage in EW-Shopp**

**Installation guide**

https://github.com/datagraft/grafterizer-2.0

**User guide**

https://github.com/datagraft/datagraft-reference/blob/master/documentation.md#transform

**API documentation**


**Software license**


**Source code repository**

https://github.com/datagraft/grafterizer-2.0

**Contact person**

nikolay.nikolov@sintef.no

### 2.2.1.2 Architecture

Grafterizer is implemented using a microservice architecture, whereby it is composed of micro-components, each contained in Docker containers, which implement its different functionalities. Figure 3 shows a high-level overview of the main functional components related to Grafterizer and a more detailed description of the individual microservices that implement the functionalities is available in Deliverable D2.4. The microservice architecture allows to avoid vendor lock-in and compose complex applications based on different technologies. It also enables decomposing applications into well-defined sub-components, thus better isolating possible faults and ensuring better scalability of the individual sub-components. The microservices themselves are deployable locally and are also available on the DataGraft portal, which has been deployed using Amazon Web Services (details on the deployment are available in deliverable D2.4). Local deployments can be
used for development purposes or for running private instances of the Grafterizer component and the DataGraft platform.

In terms of functionalities, Grafterizer's integration with DataGraft allows it to take advantage of DataGraft's platform services for user account, asset management and data hosting to store/load transformations as well as input CSV files. Furthermore, Grafterizer enables users to perform reconciliation and extension as well as schema-level annotation by providing a UI for the ASIA and ABSTAT services. The actual data preparation/transformation is done in two modes: online, using client-server communication (with a back-end transformation interpreter service), and offline, using transformation executables that are compiled live based on the transformation steps specified by the user in the Grafterizer UI.

In offline environments, Grafterizer-based transformations can be executed at scale by injecting the transformation scripts in instances of specialized Docker images that are optimized to be able to run transformations in parallel in order to transform huge amounts of data. The instances of the images can be deployed on a set of managed hosts as shown in Figure 4.
Scalable containers can be composed in more complex ways to form basic data workflows that include pre- and post-processing steps such as fetching data, splitting large files in chunks, storing transformation results in a database or others. Step composition is made possible through the use of distributed file system mounts that needs to be made available. Details about the scalable back-end (a.k.a., the Processing component) are available in deliverables D2.2 and D2.4. The scalable back-end enables a library of available steps and provides a generic template that can be extended for custom steps. When deploying transformations (or other applications) at scale, apart from the Grafterizer scalable transformation image name, users need to specify parameters, filesystem mounts as well as the number of instances that need to be deployed. The container orchestrator manages the distributed deployment of the containers that instantiate the transformation scaling images across the managed cluster. The Grafterizer tool provides a user interface to generate deployment configurations (in YAML format) for the Rancher\(^\text{12}\) container orchestrator by specifying the input parameters.

### 2.2.1.3 Functionality

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Existing</th>
<th>New (P1)</th>
<th>New (P2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data import</td>
<td>Loading CSV data directly from the user’s managed assets on the DataGraft platform, or from disk.</td>
<td></td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

\(^{12}\) [https://rancher.com/](https://rancher.com/)
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation storage and retrieval</td>
<td>Serializing and storing transformations as DataGraft assets (including their metadata and sharing rights in the platform) and loading them in the Grafterizer UI.</td>
<td>+</td>
</tr>
<tr>
<td>Suggestion-based tabular data cleaning and transformation</td>
<td>Providing users with suggestions for their choice of tabular data cleaning, including visual data profiling for data quality assessment, and semantic annotation of CSV data to RDF knowledge graphs.</td>
<td>+</td>
</tr>
<tr>
<td>Template-based RDF mapping</td>
<td>Grafterizer provides a user interface for mapping the loaded tabular data into RDF triples through the specification of mapping templates.</td>
<td>+</td>
</tr>
<tr>
<td>Scalable data transformation backend and templates</td>
<td>An architecture based on container technology and a shared file system that allows for running data transformations at scale when processing very large amounts of data in parallel.</td>
<td>+</td>
</tr>
<tr>
<td>Data export in ArangoDB JSON collections</td>
<td>A service for generating ArangoDB JSON collections that uses RDF mapping templates for structuring the output. The tool is available both as a script (to be used offline for larger amounts of data) and as a RESTful web service (used by the Grafterizer UI).</td>
<td>+</td>
</tr>
<tr>
<td>Support for load-balanced extension of very large datasets</td>
<td>The generated scripts from Grafterizer have been extended to include web service invocations to a pre-specified ASIA endpoint (possibly a set of ASIA services that are load-balanced). This functionality is used when the volumes of data exceed the amounts that are displayable in the UI of Grafterizer and must be processed offline.</td>
<td>+</td>
</tr>
</tbody>
</table>

### 2.2.2 ASIA

ASIA\(^{13}\) is a tool for the semantic enrichment of data in tabular formats. Joining tabular data that does not use the same record ids or some other identifying values is not straightforward, it requires to create associations from the table elements to a shared system of identifiers. ASIA aims to help

users in creating these links, by means of especially designed semantic reconciliation algorithms and a neat graphical interface. In this context, the linking can be performed at two different levels:

- **Schema-level linking:** linking table schema values (i.e. the header of a table) to shared vocabularies and ontologies. User-defined concepts are supported as well, though. The result of this activity is an RDF mapping suitable to be used to transform the tabular input data into a semantic format;

- **Instance-level linking:** linking data values (that is the content of the cells) to shared systems of identifiers. The result of this phase is a table with a new column with the ids of the reconciled values. Those ids can be then used to retrieve from knowledge bases that use them the information the user deems necessary to create a richer data set from the analytics standpoint.

Schema-level and instance-level links are created by ASIA as *annotations* for the table. Users can create schema-level annotation through the ASIA interface by validating suggestions about classes and properties to be used. A multi-language suggestion service exploiting ABSTAT functionalities is used to implement this functionality. However, if the user specifies a different class (or property), ASIA is able to suggest classes (or properties) that match syntactically the user’s input (autocomplete functionality). The instance-level annotations, instead, are expected to be created by ASIA automatically, as the large size of the data set often impedes to apply annotation to the values singularly; nonetheless, ASIA features specially designed tab that allows the user to interact with the candidate entities provided by the tool. The user can therefore validate, discard and change the threshold if needed.

Table annotations underpins two different data quality related functionalities of ASIA:

- **Generation of knowledge graphs from a tabular data set:** the schema-level annotations are transformed into Grafterizer data transformations to publish tabular data as a knowledge graph; data values will be used to create new instances and populate the graph.

- **Enrichment of tabular data with third-party data:** instance-level annotations are used to facilitate enrichment of user data with data retrieved from these reference knowledge graphs, often referred to as ‘core data’ in EW-Shopp lingo. Examples of those data are: GeoNames, Google GeoTargets, DBpedia, GfK products, etc.

The ASIA interface is developed as component of Grafterizer (Figure 5). The ASIA Backend is developed and maintained by UNIMIB. Suggestion and enrichment services communicate with the ASIA Backend via REST APIs.

### 2.2.2.1 Factsheet

Table 5: ASIA – Fact sheet

<table>
<thead>
<tr>
<th>ASIA</th>
<th>Functionality</th>
<th>ASIA (the name stands for Assisted Semantic Interpretation and Annotation of</th>
</tr>
</thead>
</table>

EW-Shopp  GA number: 732590  H2020-ICT-2016-2017/H2020-ICT-2016-1  |
tables) is a table-annotation tool primarily designed to use semantic approaches for the enrichment of tabular data with new information, possibly coming from third-party data sources. It supports both schema-level annotation to map a table to a (possible user-defined) ontology in such a way that an RDF-compliant transposition of the original data set can be safely generated, and instance-level annotation to reconcile table values against identifiers of entities described in existing knowledge bases or databases. Such identifiers are, therefore, added to the original table to refine the RDF-ization process as well as to fetch new pieces of data using the identifiers as access keys (data enrichment).

To support schema-level and instance-level annotation, ASIA interoperates with vocabulary suggestion services (e.g., ABSTAT full-text search, Linked Open Vocabulary), reconciliation services (e.g., Wikifier or GeoNames), data extension services (as the Weather extension service) and sameas services (e.g., the GeoNames2LAU service).

Below a list of high-level functionalities is reported; notice that we separated those implemented during the project from those already available.

More in detail, the following functionalities have been developed within the EW-Shopp project:

- **User-in-the-loop instance-level semantic annotation of tables**: the user is actively involved in the creation of instance-level annotations interacting with system, that is selecting a suitable reconciliation service, validating the entity candidates, tweaking the accuracy threshold;

- **Instance-level reconciliation services**: the reconciliation services support users in linking values in a table to identifiers of known knowledge graphs. In this category fall those services that reconcile the values of a column against a system of identifiers (reconciliation services), those that uses the reconciled ids to guide the user in the extraction of new pieces of information from a third-party data source (extension services), and sameas services, that are utilities that map different systems of identifiers to each other to make possible to reconcile once and extend from several sources;

- **RDF creation**: based on the instance-level semantic annotations, RDF mappings are generated to achieve linked knowledge graphs when transformations are executed.

Pre-existing features that are not attributable to the work done in the EW-Shopp project are:

- **Interactive schema-level semantic annotation of tables**: the user interface supports the creation of schema-level semantic annotation and is integrated into the Grafterizer tool to make the semantic annotation and
RDF transformation processes integral to the data transformation steps;

- **Schema-level vocabulary suggestions**: ASIA incorporates schema-level suggestions provided via API using ABSTAT, a knowledge graph profiling tool. It can be configured to use LOV and other terminology recommendation services like the ones;

<table>
<thead>
<tr>
<th>Usage in EW-Shopp</th>
<th>ASIA provides data enrichment services that allowed users of the EW-Shopp toolkit to reconcile (for instance) geographic toponyms to be extended with weather and event information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation guide</td>
<td><a href="https://github.com/UNIMIBInside/asia-backend">https://github.com/UNIMIBInside/asia-backend</a></td>
</tr>
<tr>
<td>User guide</td>
<td><a href="https://github.com/UNIMIBInside/asia-backend">https://github.com/UNIMIBInside/asia-backend</a></td>
</tr>
<tr>
<td>API documentation</td>
<td><a href="https://github.com/UNIMIBInside/asia-backend">https://github.com/UNIMIBInside/asia-backend</a></td>
</tr>
<tr>
<td>Software license</td>
<td>ASIA comprises different services and an heterogenous system of licenses:</td>
</tr>
<tr>
<td></td>
<td>1. The graphical interface inherits from Grafterizer 2.0 the licence, that is Eclipse Public License 1.0</td>
</tr>
<tr>
<td></td>
<td>2. The ASIA-backend and the ASIA-MAS services are licensed under the GNU Affero General Public License v3.0 license. The Conciliator service adopts the GNU General Public License v3.0 license.</td>
</tr>
<tr>
<td>Source code repository</td>
<td><a href="https://github.com/UNIMIBInside/asia-backend">https://github.com/UNIMIBInside/asia-backend</a></td>
</tr>
<tr>
<td>Contact person</td>
<td><a href="mailto:matteo.palmonari@unimib.it">matteo.palmonari@unimib.it</a></td>
</tr>
</tbody>
</table>

### 2.2.2.2 Architecture

In Figure 6 the two main logical components of ASIA are depicted and the relationship with ABSTAT (a component of EW-Shopp toolkit that analyzes and summarizes knowledge bases in RDF format to provide vocabulary suggestions, see Section 2.2.3):

- **ASIA Backend**, which contains the logic related to the annotation suggestion service and that is responsible for the intercommunication with other services (like, but not limited to, ABSTAT) to provide annotation suggestions and autocomplete service for the schema-linking functionality as well as reconciliation and extension features for the instance linking.
- **ASIA Frontend**, which is the application frontend, integrated within Grafterizer. Figure 5 depicts the ASIA’s tabular annotation panel within Grafterizer’s user interface.

![Figure 5: ASIA integrated in Grafterizer](image)

A diagram illustrating the services involved in the instance-linking process is presented in Figure 7. In particular, the following components are identifiable:

1. ASIA frontend interacts with the ASIA backend sending the data to be reconciled and enriched together with a possible context.
2. ASIA backend is responsible for calling and orchestrating the appropriate reconciliation and extension services among those available. The extension can also be made using EW-Shopp core services, i.e., weather and events, which use date and position references as reference identifiers.
3. Reconciliation and extension services. In the figure, only a subset of the services available is depicted for space reasons.

![Figure 7. The interlinking service architecture.](image)

### 2.2.2.3 Functionality

Table 6: ASIA - Summary of features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Existing</th>
<th>New (P1)</th>
<th>New (P2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schema-level semantic annotation (schema-level linking)</strong></td>
<td>ASIA provides suggestions to the user on how to annotate (type and property) a column. In both cases ASIA exploits LOV and ABSTAT as autocomplete services.</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td><strong>Instance-level semantic annotation: reconciliation</strong></td>
<td>Interface to use reconciliation services that are compliant with OpenRefine APIs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIAF, ORCID, Open Library, Solr</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wikidata</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wikifier</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GeoNames</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GeoTarget</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GFK products</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword2Category</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Instance-level semantic annotation: data extension</strong></td>
<td>Interface to use data extension services that are largely compliant with OpenRefine APIs (supported services are: ECMWF Weather Enrichment, GeoNames, Events).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wikidata</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECMWF weather enrichment</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoNames</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GeoNames2LAU sameas service</strong></td>
<td>Service that maps GeoNames identifiers to LAU codes (administration level 4) to enrich GeoNames data.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Transformation (RDF-ization and enrichment)</strong></td>
<td>Annotations are converted into auto-generated data transformations (in Clojure) that can be applied to generate RDF data or enrich the input table.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Given a table, ASIA provides an interface that guides the user through the annotation task using a column-wise approach. This approach allows users to define annotations for each column in such a way that the annotations encode enough information to automatically generate the transformations required for RDF-ization.

**Schema-level linking functionalities.** ASIA provides schema matching functionalities both in automated and manual model. In the automated mode ASIA processes the header of each column to identify meaningful keywords. Space trimming, camelCase and underscore splitting are an example of the operations performed on the headers. Such keywords are used by the ASIA backend to invoke external services like LOV and ABSTAT to obtain a list of candidate annotations for each column. Both LOV and ABSTAT returns the candidates sorted on a likelihood criterion. ASIA selects the most likely annotation. The user is always responsible for validating the proposed annotation. Unvalidated annotations, in fact, will not be taken into account when creating the RDF of the table.

In many cases, however, the header is too cryptic and may not carry enough information to be exploited to get automatic annotation. For such cases, ASIA provides a syntactic autocomplete

---

14 Transformations based on instance-level annotations are the result of activities in EW-Shopp, while transformations based on schema-level annotations are preexisting.
function, that is, an online tool capable to provide recommendations based on what the user is typing.

**Instance-level interlinking functionalities.** ASIA also provides instance-level interlinking functionalities using different existing reconciliation and data extension services:

- Data reconciliation services available today in ASIA:
  - Wikidata (from OpenRefine);
  - Wikifier (cross-lingual service, see D2.2 “Cross-lingual/Multilingual Data Management Approach for Structured and Unstructured Data”);
  - GeoNames (native ASIA reconciliator).
- Data extension services available today in ASIA:
  - ECMWF Weather Enrichment (ASIA extension service);
  - GeoNames (ASIA extension service);
  - Wikidata (pre-existing service from the OpenRefine ecosystem);
  - GeoNames2LAU (ASIA sameas service).

### 2.2.3 ABSTAT

ABSTAT is a Knowledge graph profiling tool. Knowledge repositories may potentially contain millions of facts describing entities such as people, places, events, films, etc. Such knowledge in the Semantic Web is encoded into graphs represented using the RDF data model. One example of knowledge bases is the Linked Open Data (LOD) Cloud with roughly 1,184 data sets belonging to different domains such as geographic, life science, publications, media, cross-domain, etc. There are many data sets that are of a high quality in the LOD Cloud; however, there are also many data sets which are extracted from unstructured or semi-structured information being vulnerable for quality issues. In the state-of-the-art, many metrics, methodologies and approaches have been proposed to capture quality issues in linked data sets, some of them are implemented in ABSTAT.

Given an RDF data set and, optionally, an ontology (used in the data set), ABSTAT\(^\text{15}\) computes a summary that provides an abstract but complete description of the data set content. Summaries are published and made accessible via a web interface and an API, in such a way that the information they contain can be consumed by human users and machines. ABSTAT makes also use of a minimalization mechanism to keep summaries complete but as small as possible.

Jointly, ASIA and ABSTAT work to implement the schema-level linking (also referred to schema alignment) functionality. The main purpose of this feature is to increase the level of automation of

\(^{15}\) [http://backend.abstat.disco.unimib.it/](http://backend.abstat.disco.unimib.it/)
the creation and publishing process of the data set in RDF format, but clearly it also contributes to improving the final quality of the Knowledge Base created by suggesting to the user the most appropriate annotations and thus reducing the risk of generating incorrect RDF mapping.

It should be noted that although ABSTAT is used by ASIA to simplify the process of publishing data in semantic format and to improve the quality of the final knowledge graph (it is part of the quality services of EW-Shopp), its development pre-dates the project and has not been substantially improved by EW-Shopp. For this reason, in this section and in the following ones, we limit ourselves to briefly describing it, omitting to provide an analytical description of the architecture and functionalities.

### 2.2.3.1 Factsheet

Table 7: ABSTAT – Fact sheet

<table>
<thead>
<tr>
<th>ABSTAT</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTAT is a tool to create, manage and serve semantic profiles of knowledge graphs. Such semantic profiles describe structural properties and vocabulary of knowledge graphs. Several tasks can benefit or build upon semantic profiles; examples are data exploration, data quality, vocabulary recommendation, schema alignment. Access to information stored in profiles is accessible to users, by means of Web-based interfaces, as well as to machines, by means of suitable APIs. ABSTAT’s main features (all available before the project):</td>
</tr>
<tr>
<td></td>
<td>• Core profiling algorithms: algorithms for the extraction of patterns and the computation of pattern-specific statistics;</td>
</tr>
<tr>
<td></td>
<td>• ABSTAT GUI and API: a user-oriented Web application and API’s to control the profiling process and compute, store and manage profiles;</td>
</tr>
<tr>
<td></td>
<td>• ABSTAT Distributed implementation: the distributed architecture used to control, compute, and store profiles, as well as to represent and share them via the HTTP protocol. In particular, the new architecture uses NoSQL database to store profiles for structured access and ElasticSearch for full-text search over the profiles;</td>
</tr>
<tr>
<td></td>
<td>• ABSTAT ShaclGenerator: The ShaclGenerator is a feature integrated into ABSTAT which, given as input a file containing the heuristic cardinality, for each minimal pattern generates as output a shape containing the direct and inverse cardinality constraints for the pattern under investigation. Once such shapes are validated, in the user interface users can distinguish two messages (Valid, Invalid). A new API is made available to support users to view the triples that do not satisfy the constraints in the shape.</td>
</tr>
<tr>
<td><strong>Usage in EW-Shopp</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ABSTAT provides summaries of a set of common knowledge graphs to be used by ASIA in order to supply suggestion for the schema-level linking. In addition, ABSTAT can be used to spot quality issues on the final published data.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Installation guide</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A guide to installing ABSTAT can be found in the Readme.me file in the ABSTAT repository on Bitbucket (<a href="https://bitbucket.org/disco_unimib/abstat">https://bitbucket.org/disco_unimib/abstat</a>).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>User guide</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://bitbucket.org/disco_unimib/abstat">https://bitbucket.org/disco_unimib/abstat</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>API documentation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://backend.abstat.disco.unimib.it/apis">http://backend.abstat.disco.unimib.it/apis</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Software license</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.gnu.org/licenses/agpl-3.0.html">https://www.gnu.org/licenses/agpl-3.0.html</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Source code repository</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://bitbucket.org/disco_unimib/abstat-distributed">https://bitbucket.org/disco_unimib/abstat-distributed</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Contact person</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:matteo.palmonari@unimib.it">matteo.palmonari@unimib.it</a></td>
<td></td>
</tr>
</tbody>
</table>

### 2.3 Key Updates Since the Previous Release

The purpose of this section is to identify, and present updates and new features brought to components after M18. In fact, much work has been done to integrate ASIA and Grafterizer at many different levels. Among other things it is worth mentioning the changes made to the Grafterizer internal data model in order to integrate the reconciliation and enrichment functionalities and the change to the pipeline execution workflow which is now partly performed in part by Graftwerk and partly by ASIA. Finally, Jarfter has been modified to create jar executables capable of invoking ASIA at runtime, with the advantage of being able to redo the reconciliation and enrichment even on data not encountered before.

The rest of this section is devoted to present some novel functionalities added to ASIA since previous release of the toolkit, while, a comparative assessment of the enrichment component has been performed and presented (in Appendix A) to accompany and complement that presented in D3.4.

### 2.3.1 Updates to ASIA

As far as interlinking services are concerned, the major updates compared to D3.2 refer to the creation and integration of the keyword2category service (which uses FastText\(^\text{16}\)'s pre-trained

\(^{16}\text{https://fasttext.cc/}\)
embeddings to associate a Google marketing category in output with an input keyword, see Chapter 4 for more details) and the event enrichment service (which interfaces with Event Registry\textsuperscript{17} to allow the user to extend the work table with additional event data).

In particular, the core of the new enrichment service called keyword2category here, consists of an algorithm that exploits the distributional semantics (using pre-trained models and made available by Facebook) to identify the categories of Google closest semantically. This algorithm, written in Python could not be directly integrated in ASIA, whose ecosystem of services is realized using the Java language, for this reason we have created a service REST based on Flask, containerized and freely available\textsuperscript{18}. This service is invoked by ASIA-Backend through Conciliator, which provides for the service a Java API compatible with OpenRefine.

Figure 8 and Figure 9 show, respectively, the reconciliation widget for the keyword2category service in which the user can interact by changing the threshold and possibly overriding the output of the service, and the column reconciled with the id of the categories associated with the keywords.

\textbf{Figure 8: ASIA’s keyword2Category reconciliation widget}

\textsuperscript{17} \url{http://eventregistry.org/}
\textsuperscript{18} \url{https://github.com/UNIMIBInside/ew-shopp-public/tree/master/keyword_clustering}
The other feature added to ASIA after M18 is the ability to manage events, whether they are custom events for which an ontology and an API\textsuperscript{19} has been defined specifically or whether they are general events or media coverage events (presented in detail in Chapter 3) provided by the Event Registry service.

For reasons of scalability and following the same choice made for the management of weather data, it was decided that ASIA should mainly access the events contained in a local repository (ArangoDB). This greatly reduces the impact of network latency on the enrichment process. For this reason, an application has been developed that is able to communicate with the specific API of the events and download them in json-ld format. This service, which is currently part of the platform is released under an open source license and available on github\textsuperscript{20} and Docker Hub\textsuperscript{21}.

In order to make ASIA functional and locally deployable and to avoid requiring the user to implement an automatic and scheduled download system for events.

\textsuperscript{19} \url{https://app.swaggerhub.com/apis/EW-Shopp/EW-Shopp_Event_API/2.2.0}

\textsuperscript{20} \url{https://github.com/UNIMIBInside/ews_events}

\textsuperscript{21} \url{https://hub.docker.com/r/miciav/eventsdownloader}
Chapter 3 Data Analytics

The core of the EW-Shopp analytics is the Event and Weather Analytics Toolset which has been developed for use over all business cases and was used by most of the pilots. This toolset was described in detail in deliverable D3.2 and its API has changed little since then, as the functionality it provides has been required by the pilots. This functionality pertains to the ingestion of the data, using it to build predictive models and then use them to produce predictions. The changes made since the pilots were minor and of a such technical nature, we believe they are better placed in the online documentation\(^\text{22}\) than listed here. Note that this does not mean the models created in the business cases or their use in the business process hasn’t changed. Some have changed quite significantly, but these changes were implemented within the existing API which was designed to be general enough to allow for exactly these kinds of adaptations and changes.

In this chapter we first present the factsheet with the basic information on the Event and Weather Analytics Toolset in section 3.1. Then we focus on the two main high-level extensions completed in the second half of the project. The first one is the keyword clustering tool (section 3.2) and the support for extracting features from events covered in the media (section 3.3). These extensions, though in the scope of the project primarily motivated by the JOT business case, are very general in purpose and can be used in various business scenarios.

Together these three tools represent the main services for exploitation for JSI as described in D5.5. With the exception of the keyword clustering tool, which due to its close dependency on external libraries was implemented in the Python programming language, all the codebase is based on that of the QMiner platform. This allows them to be used together with little overhead, even though they can just as easily be used each on their own and is the reason the analytics code is occasionally referred to in some documents and presentations as simply “QMiner”, as its greatest unifying feature. Finally, note that deliverable D2.4 (as its predecessor D2.2) focuses mostly on the role of the event and weather toolset in the EW-Shopp platform as this is the part most generally used over all the business cases. The points where the other two tools can be plugged in are detailed in chapter 2.3.1 of D2.4.

3.1 Event and Weather Analytics Toolset

As explained in the introduction of this chapter above, the detailed description of this toolset can be found in deliverable D3.2. For completeness and clarity, we here present just the factsheet for this toolset in Table 8 below.

Table 8: Event and Weather Analytics Toolset – Fact sheet

\(^{22}\text{https://github.com/JozefStefanInstitute/ew-shopp-public/tree/master/analytics}\)
### Event and Weather Analytics Toolset

#### Functionality

The Event and Weather Analytics Toolset contains all the functionality needed to ingest data for analysis, process the data to produce predictive models and then use these models to produce predictions. Data ingestion is possible from different sources and formats including csv files and SQL as well as non-SQL databases. For the modelling step the entirety of the QMiner modelling library is available. Finally, a simple and efficient REST server is available for querying the models and obtaining predictions.

#### Usage in EW-Shopp

The toolset was used in several business cases – namely Ceneje.si, Big Bang, BrowseTel and JOT. The models produced by it have tackled very different tasks from predicting the number of people visiting stores to identifying the optimal day for releasing a marketing campaign. The production instances of the modelling and prediction services were deployed on business partner infrastructure.

#### Installation guide

https://github.com/JozefStefanInstitute/ew-shopp-public/blob/master/analytics/README.md

#### User guide

https://github.com/JozefStefanInstitute/ew-shopp-public/blob/master/analytics/README.md

#### API documentation

https://github.com/JozefStefanInstitute/ew-shopp-public/blob/master/analytics/README.md

#### Software license

https://github.com/qminer/qminer/blob/master/LICENSE

#### Source code repository

https://github.com/JozefStefanInstitute/ew-shopp-public

#### Contact person

aljaz.kosmerlj@ijs.si

### 3.2 Keyword Clustering Tool

The keyword clustering tool was developed for the JOT business case. As an advertising agency focusing on online ads, JOT processes vast amounts of internet traffic data. This data, bought from various online search engines (primarily Google), records how many impressions were generated by individual keywords. An impression is generated when a user searches for a keyword that is part of an active campaign. Note that we use the term “keyword” as it is used in this field to denote multi-word phrases (e.g. “hamburger”, “holidays in Portugal”, “football game”).
To optimize its operation JOT aims to model the level of impressions for keywords using contextual features (i.e. weather and events). As there are millions of different keywords that behave differently based on their meaning. To alleviate this problem, a method for collecting the keywords into meaningful groups that would allow group-level modelling is needed. There exists a hierarchical category system for the keywords, but although the categories are known, the categories of individual keywords are not.

### 3.2.1 Factsheet

An overview of the basic information regarding the tool is presented in Table 9.

**Table 9: Keyword Clustering Tool – Fact sheet**

<table>
<thead>
<tr>
<th><strong>Keyword Clustering Tool</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
</tr>
<tr>
<td><strong>Usage in EW-Shopp</strong></td>
</tr>
<tr>
<td><strong>Software license</strong></td>
</tr>
<tr>
<td><strong>Source code repository</strong></td>
</tr>
<tr>
<td><strong>Contact person</strong></td>
</tr>
</tbody>
</table>
### 3.2.2 Approach

The problem can be formulated as a clustering\(^{23}\) task. To measure the semantic similarity between the keywords an embedding approach is used. An embedding of a word is a mapping of the word to a point in a (relatively) low-dimension space in such a way that semantically related words are close together and unrelated far apart. The embedding mappings need to be trained on large corpora of text to be effective. Fortunately, pre-computed mappings can be obtained online, having been built and made available by research institutions and companies (e.g. Google, Facebook).

Embeddings are computed for individual words. To compute a semantic representation of keywords, which can be comprised of multiple words, we need to aggregate the embeddings of the words in the keyword. We use the Smooth Inverse Frequency (SIF) approach\(^ {24}\) for building sentence embeddings, where the embeddings of the words are weighted by the inverse of their probability and summed up and then a correction is done by removing the main component of the SVD decomposition on the matrix of the words’ embeddings. The full details of this algorithm exceed this overview. Suffice to say, that this approach can adapt the embeddings to the vocabulary of the keywords we are using. For example, the word “and” is very common in the keywords and shouldn’t contribute to the estimation if two are similar as it carries little relevant meaning.

Having the embeddings, computing the semantic similarity between the keywords is obtained by computing the (cosine) distance between the keywords’ embeddings. In simpler terms, we can calculate a number which tells us how related any pair of keywords is. This allows us to run standard clustering algorithms. Since we have category names we can use them as centroids of the clusters by embedding them using the same approach and collecting the closest keywords into the cluster.

### 3.2.3 Implementation

For computation of the embeddings we use FastText\(^ {25}\), an open-source library for text representation learning created by Facebook. FastText offers pre-trained word vectors for 157 languages – including German and Spanish, which are used in the JOT dataset. In our work we focused on German, but the same process could be repeated for any supported language. Recently, FastText has also released aligned word embeddings for 44 languages, offering the possibility to make the models cross-lingual.

For computation of the keyword embeddings from the word embeddings we use the implementation of the SIF aggregation approach provided by the authors of on GitHub\(^ {26}\).

---


\(^{24}\) Arora, S., Liang, Y. and Ma, T., 2016. A simple but tough-to-beat baseline for sentence embeddings.

\(^{25}\) [https://fasttext.cc/](https://fasttext.cc/)

\(^{26}\) [https://github.com/PrincetonML/SIF](https://github.com/PrincetonML/SIF)
The list of 3180 Google category names was provided by JOT. The categories are organized into a hierarchy and for our work we used the categories up to depth 3. An example of the three-level structure is:

```
/Apparel
/Apparel/Apparel Accessories
/Apparel/Apparel Accessories/Bags & Packs
```

Category names were provided in English. Since the keywords we use are in German, we need to use the embeddings for the German language. This is why the categories were translated into German using Google Translate through the googletrans library\(^{27}\). Since the category names are typically common and clear words, the translation is of satisfactory quality, which was confirmed with manual inspection.

Having the keywords and category names both embedded into the same space we use cosine distance\(^{28}\) to find the closest N keywords for each category name (parameter N is set to 1000 in our experiments so far). Since it is not problematic for the use-case, we do not enforce disjoint clusters and the same keyword can be in several.

### 3.2.4 Workflow

The API needs to support three main operations:

a) Fitting the embedding parameters for keyword distances,

b) Finding the clusters of most relevant keywords for each category,

c) Assigning new keywords to the categories.

**a) Fitting the embedding parameters for keyword distances:** This is the initial operation done to fit the semantic similarity measure to the vocabulary we are using. It is performed once or occasionally for each language when there is a major change in vocabulary of the keywords. It computes a set of internal parameters for the embedding process optimised to the keyword set.

The input into this operation consists of:

- A (preferably large) representative set of keywords in the target language,
- The FastText model file for the target language obtained from the official FastText repository

and the output is a set of embedding parameters which is used as input for all subsequent operations in this vocabulary space.

**b) Finding the clusters of most relevant keywords for each category:** The main purpose of the clustering tool is to find relevant keywords for each category, so the information about their


\(^{28}\) [https://en.wikipedia.org/wiki/Cosine_similarity](https://en.wikipedia.org/wiki/Cosine_similarity)
impressions can be aggregated together to obtain a more representative category-level signal. This operation computes for each category name the distance to all keywords and assigns only the top N closest to the category. Here N is the parameter of the operation (typical value is N=1000). Note that the only purpose is to characterise the categories, so some keywords may not be assigned to any category and some could possibly be assigned to more than one. Typically, the set of input keywords is the same as the one used for fitting the embedding parameters. Similar to the fitting operation this operation is also performed once or occasionally for each language when there is a major change in vocabulary of the keywords.

The input into this operation consists of:

- The FastText model file for the target language obtained from the official FastText repository,
- The embedder parameters fitted to the keyword vocabulary,
- The category names in the target language,
- A set of keywords we want to use to characterise the categories,
- The number, N, of how many closest keywords we want to consider per category,

and the output is for each keyword a list of categories the keyword is among the top N closest or ‘none’ if the keyword is not close enough to any category.

c) Assigning new keywords to the categories: The category clusters are used for modelling impressions and the predictions of their dynamics are created at the category cluster level. When a prediction for a keyword is needed, it must first be assigned to a cluster by its semantics – i.e. the cluster of the category name to which the keyword is closest. This operation is performed for each marketing campaign when a prediction needs to be generated for a set of new keywords. To offer more insight into the keywords and operational flexibility, the closest K categories are returned (typical value is K=3). This allows the campaign manager to use alternative categories based on expert opinion.

The input into this operation is almost identical to that of the previous one and consists of:

- The FastText model file for the target language obtained from the official FastText repository,
- The embedder parameters fitted to the keyword vocabulary,
- The category names in the target language,
- A set of keywords we want to assign to the categories,
- The number, K, of how many closest categories we want per keyword,

and the output is for each keyword a list of K categories the keyword is closest to.

### 3.2.5 Usage
The API is used through its command-line interface. Each of the three main operations is executed using a specific command. It is coded in Python programming language and consists of the following scripts:

- **embedder.py**: Used for the operation a).
- **categoriser.py**: Used for the operations b) (invoked by `relevance_to_category` argument) and c) (invoked by `categorise_keywords` argument).

**a) Fitting the embedding parameters for keyword distances**: This operation is invoked using the following command:

```
$ python embedder.py build <fasttext_bin> <keywords_csv> <embedder_json>
```

The `<fasttext_bin>` is a path to the file containing pre-trained word vectors in binary format, downloaded from fasttext webpage. For Spanish language this file is `cc.es.300.bin`. The `<keywords_csv>` is a csv file containing the keywords in the target language in one of the columns. The name of the keywords’ column can be arbitrary and is specified via the optional `keywords_column` argument. The output of this operation is a json file stored in `<embedder_json>` containing the parameters of the language-specific embedding model.

The memory footprint of this operation scales linearly with the number of keywords, as the initial 300-dimensional embedding of each keyword has to be calculated in order to estimate the required embedding parameters. We have observed that estimating those parameters on a smaller (random) subsample of provided keywords is indeed a very good approximation. As the number of keywords can be as large as 25 million in our case, we sample a smaller subset of keywords to keep the memory footprint manageable. The size of this subset is specified by the optional argument `sample` which is set to 1 million by default.

The operation can be configured using the following optional arguments:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keywords_delimiter</td>
<td>Char</td>
<td>Delimiter used in the <code>&lt;keywords_csv&gt;</code> file. (default: “,”)</td>
</tr>
<tr>
<td>keywords_column</td>
<td>String</td>
<td>Name of column containing keywords in the <code>&lt;keywords_csv&gt;</code> file. (default: “Keyword”)</td>
</tr>
<tr>
<td>sample</td>
<td>Integer</td>
<td>Size of the random sample of keywords. (default: 1000000)</td>
</tr>
</tbody>
</table>

**b) Finding the clusters of most relevant keywords for each category**: This operation is invoked using the following command:

```
$ python categoriser.py relevance_to_category <fasttext_bin> <embedder_json> <categories_csv> <keywords_csv> <output_csv>
```

The `<fasttext_bin>` is (same as in the operation a) a path to the file containing pre-trained word vectors in binary format, downloaded from fasttext webpage. The `<embedder_json>` is the json file containing the parameters of the embedding model which are calculated as the result of the
The `<categories_csv>` is a path to the csv file containing the categories in the target language in one of the columns, specified via the `categories_column` optional argument, and the ids of the categories in another column, specified via the `categories_id_column` optional argument. The `<keywords_csv>` is a path to the csv file containing the keywords in the target language, usually the same as in the operation a, in the column specified with the `keywords_column` optional argument. The output of this operation is stored in the tab-separated csv file `<output_csv>` with the following format:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword1</td>
<td>Category1(Id1), Category2(Id2),...</td>
</tr>
</tbody>
</table>

where for each keyword from `<keywords_csv>` all the categories, in which this keyword is among the top N closest to the category, are listed together with their ids. If there are no such categories a single string “none” is stored under the Categories column. The value of N can be specified via the `n_keywords` optional argument and is set to 1000 by default.

The memory footprint of this operation is dominated by the size of the embedding model and is practically independent of the number of keywords. It depends on the size of the result, which is still insignificant for any reasonable number of categories, which is around 3000 in the JOT case, and the value of N, which is not expected to change.

The operation can be configured using the following optional arguments:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_keywords</td>
<td>Integer</td>
<td>Number of top N closest keywords for each category. (default: 1000)</td>
</tr>
<tr>
<td>categories_delimiter</td>
<td>Char</td>
<td>Delimiter used in the <code>&lt;categories_csv&gt;</code> file. (default: &quot;,&quot;)</td>
</tr>
<tr>
<td>categories_column</td>
<td>String</td>
<td>Name of column containing categories in the <code>&lt;categories_csv&gt;</code> file. (default: “Category”)</td>
</tr>
<tr>
<td>categories_id_column</td>
<td>String</td>
<td>Name of column containing category ids in the <code>&lt;categories_csv&gt;</code> file. (default: “CategoryID”)</td>
</tr>
<tr>
<td>keywords_delimiter</td>
<td>Char</td>
<td>Delimiter used in the <code>&lt;keywords_csv&gt;</code> file. (default: &quot;,&quot;)</td>
</tr>
<tr>
<td>keywords_column</td>
<td>String</td>
<td>Name of column containing keywords in the <code>&lt;keywords_csv&gt;</code> csv file. (default: “Keyword”)</td>
</tr>
</tbody>
</table>

**c) Assigning new keywords to the categories:** This operation is invoked using the following command:

$ python categoriser.py categorise_keywords <fasttext_bin> <embedder_json> <categories_csv> <new_keywords_csv> <output_csv>
The `<fasttext_bin>` is (same as in the operations a and b) a path to the file containing pre-trained word vectors in binary format, downloaded from fasttext webpage. The `<embedder_json>` is the json file containing the parameters of the embedding model which are calculated as the result of the operation a. The `<categories_csv>` is a path to the csv file containing the categories in the target language in one of the columns, specified via the categories_column optional argument, and the ids of the categories in another column, specified via the categories_id_column optional argument. The `<new_keywords_csv>` is a path to the csv file containing the possibly previously unseen keywords in the target language in the column specified with the keywords_column optional argument. The output of this operation is stored in the comma-separated csv file `<output_csv>` with the following format:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Category1_id</th>
<th>Category1</th>
<th>Category1_distance</th>
<th>...</th>
<th>CategoryK_id</th>
<th>CategoryK</th>
<th>CategoryK_distance</th>
</tr>
</thead>
</table>

For each keyword the ids, category names and distances to this keyword for the closest K categories are stored. where Category1 is the closest and CategoryK is the k-th closest category. The value of K can be specified via the n_categories optional argument and is set to 3 by default.

The memory footprint of this operation is dominated by the size of the embedding model. It also depends on the size of the result, which is insignificant for any reasonable value of K, which is set to 3 in JOT case and is not expected to increase above 10. The result is independent for each keyword so in case of large number of keywords they can be easily batched and processed sequentially thus avoiding the need to keep them all in working memory.

The operation can be configured using the following optional arguments:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_categories</td>
<td>Integer</td>
<td>Number of closest K categories to return. (default: 1000)</td>
</tr>
<tr>
<td>categories_delimiter</td>
<td>Char</td>
<td>Delimiter used in the <code>&lt;categories_csv&gt;</code> file. (default: “,”)</td>
</tr>
<tr>
<td>categories_column</td>
<td>String</td>
<td>Name of column containing categories in the <code>&lt;categories_csv&gt;</code> file. (default: “Category”)</td>
</tr>
<tr>
<td>categories_id_column</td>
<td>String</td>
<td>Name of column containing category ids in the <code>&lt;categories_csv&gt;</code> file. (default: “CategoryID”)</td>
</tr>
<tr>
<td>keywords_delimiter</td>
<td>Char</td>
<td>Delimiter used in the <code>&lt;keywords_csv&gt;</code> file. (default: “,”)</td>
</tr>
<tr>
<td>keywords_column</td>
<td>String</td>
<td>Name of column containing keywords in the <code>&lt;keywords_csv&gt;</code> file. (default: “Keyword”)</td>
</tr>
</tbody>
</table>

### 3.2.6 JOT Case Optimization

The tool was initially designed to handle at most 1 million keywords and had to be optimized in order to scale to the larger JOT German dataset containing more than 3000 categories and more
than 25 million keywords. The biggest constraint was the memory, which was heavily influenced by the number of keywords before the optimization.

In the operation a, the 300-dimensional embedding for each keyword was stored in memory to calculate the SVD decomposition required by the algorithm. This was feasible for 1 million keywords but became untraceable with 25 million keywords. We have optimised this step by observing that random sampling 1 million keywords provided a good enough approximation.

Obtaining the results for the operation b requires calculating a cosine distances between each keyword-category pair, which can be simultaneously done for multiple pairs via matrix multiplication. To calculate the distance both the keyword embedding and category embedding have to be calculated. Storing the embedding for each of the 25 million keywords in memory is again infeasible, so the raw keywords and categories are stored in batches of size at most 4000 and the embeddings are calculated exactly when needed. This ensures a constant memory footprint, independent of the number of keywords and categories, as at most 8000 embeddings are stored in memory at any given time.

### 3.3 Media Attention Tool

Experiments performed during pilot deployment in the first half of the project (see D4.2) showed that information about the events covered by the media contributes little to model performance of most business cases in e-commerce, retail and customer care. Subsequent experiments on the JOT business case have shown that this data can help in the marketing domain, where information on media attention can offer insight into web traffic.

To extract information about media attention about certain topics and concepts we used information obtained from the Event Registry media monitoring platform. A set of tools was prepared for computation of features from this data, which will be presented in this section.

#### 3.3.1 Factsheet

An overview of the basic information regarding the tool is presented in Table 10

<table>
<thead>
<tr>
<th>Media Attention Tool</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The media attention tool collects data regarding the level of media content (in number of articles, events and date mentions) focusing on some given concept during a specified time period. The data is obtained by querying the Event Registry service and then distilling the information into the key features relevant for</td>
</tr>
</tbody>
</table>

---

3.3.2 Media Attention Features

After using the keyword clustering tool, we have a set of categories that can be enriched using the events' and articles' data from the Event Registry. We use features from events and articles at the same time. An event is the cluster of articles that are about the same event. Therefore, an article is not always part of an event. To better capture media attention, we need to query both signals—from articles and events.

To query the Event Registry, we must handpick Wikipedia concepts that are suited to a specific category. We also need to provide the start and the end date of the interesting date range and list of interesting regions. These are usually provided on the country-level, to get a strong enough signal. The signal of media attention is usually not strong enough if the granularity of regions is too fine.

The media attention features can be separated into four groups—signal from the past articles, past events, future articles and future events. In the past signal, we mostly count events and articles for a particular concept in the past week, and in the future signal, we look for future date mentions that might indicate events in the nearby future.
Thereby, we calculate the following media attention features specific to a particular date, region and Wikipedia concept:

- **Past signal:**
  - ArticlesCounts: The number of articles from yesterday.
  - ArticlesCountsPastMax: Maximum number of articles per day in the past 7 days.
  - ArticlesCountsPastAvg: Average number of articles per day in the past 7 days.
  - ArticlesCountsPastMin: Minimum number of articles per day in the past 7 days.
  - EventsCounts: The number of events from yesterday.
  - EventsCountsPastMax: Maximum number of events per day in the past 7 days.
  - EventsCountsPastMin: Minimum number of events per day in the past 7 days.
  - EventsCountsPastAvg: Average number of events per day in the past 7 days.

- **Future signal:**
  - MentionedArticlesCurrent: The number of times yesterday is mentioned in the past articles.
  - MentionedEventsCurrent: The number of times yesterday is mentioned in past events.
  - MentionedArticlesFuture: The number of any future dates in yesterday’s articles.
  - MentionedEventsFuture: The number of any future dates in yesterday’s events.
  - MentionedArticlesFuture<n>: The number of mentions in yesterday’s articles of the specific future date (n days after today).
  - MentionedEventsFuture<n>: The number of mentions in yesterday’s events of the specific future date (n days after today).

Note that amongst the media attention features, there are no features that would use information obtained about the current day. Values for yesterday are used instead. This is because we do not yet know the full media coverage for the current day, since articles are still being published. Event Registry frequently updates the events and articles for the current day and a more stable media attention signal is obtained from events and articles that happened further in the past. Using the data for yesterday is still useful for gauging the level of current media attention, with the number of dates’ mentions for future dates indicating how likely is this attention to persist in the coming days.

The main idea behind the past signal is that approaching some relevant event, more and more articles and events should appear. Therefore, the signal should get stronger. On the other hand, the future signal checks if a particular date we are interested in is mentioned in the past articles or/and events. If it is, it might indicate that something relevant is going to happen on that day. We also check how many times yesterday was mentioned in the past articles. This gives us a measure of the level of media attention for the current time. Yesterday is used to ensure a full day’s worth of data. If the count is high enough, there is also a good chance for some upcoming event to be relevant to that concept.

---

The data for yesterday is used as we need a full day’s worth of articles. Using the value for the current day, for which articles are still being published, would confuse the model. This is elaborated on in the continuation of the deliverable.
Media attention features can also be easily added and updated with minimal effort and effect on the user. The API as a whole won’t change; the user would have to provide only the selection of new features if he so chooses.

### 3.3.3 Usage

The Media Attention Features API is divided into two steps. First, media attention feature transformer, which queries the Event Registry and builds features using the retrieved data. Second, feature selection, to select a subset of all calculated media attention features and use them during model building. As explained in the introduction in section 4.1, the analytics tool has changed little since D3.2 and the API’s full description is placed in the online documentation.

Automation of mapping between Wikipedia concepts and categories is not trivial. There can exist many Wikipedia concepts for a particular category, and some might be more relevant than the others. Therefore, mapping between the Wikipedia concepts and categories is done manually. The Event Registry API does provide functionality to find most likely concepts for given keywords\(^{31}\), which might provide a cheap method to get relevant concepts, but we cannot guarantee a good match will be found and assessing its reliability would require significant manual evaluation we did not have time or resources to perform.

#### 3.3.3.1 Media attention feature transformer

The transformer is specified in a configuration JSON which is given to the modelling pipeline.

```json
{
    "module": "events_features",
    "params": {
        "download": bool,
        "input_db": str,
        "output_db": str,
        "clean_db": bool,
        "queries": [
            {
                "keyword": str,
                "locations": [str, str, ...],
                "start_date": str,
                "end_date": str,
                "event_feature_id": str
            },
            ...
        ]
    }
}
```

Parameter description:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
</table>

### Download

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>download</td>
<td>Boolean</td>
<td>Yes</td>
<td>Boolean to query the Event Registry. If false, previously downloaded information is used to calculate media attention features.</td>
</tr>
</tbody>
</table>

### Input Database

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_db</td>
<td>String</td>
<td>Yes</td>
<td>Location of the QMiner database containing the raw media attention data from the Event Registry.</td>
</tr>
</tbody>
</table>

### Output Database

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_db</td>
<td>String</td>
<td>Yes</td>
<td>Location of the QMiner database where features are stored.</td>
</tr>
</tbody>
</table>

### Clean Database

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean_db</td>
<td>Boolean</td>
<td>Yes</td>
<td>Removes existing QMiner database (output_db) storing media attention features and creates new.</td>
</tr>
</tbody>
</table>

### Queries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queries</td>
<td>List</td>
<td>Yes</td>
<td>List of queries to be executed.</td>
</tr>
</tbody>
</table>

Considering that the querying the Event Registry is a slow and computationally expensive process, we build QMiner database on the local machine to store all raw information retrieved from the Event Registry. Using the flag "download" the process can execute downloading of new data from the Event Registry or skip it if the data is available locally.

Parameter description of a query:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>String</td>
<td>Yes</td>
<td>Interested category, Wikipedia concept, matching the cluster centroid.</td>
</tr>
<tr>
<td>locations</td>
<td>List</td>
<td>Yes</td>
<td>List of source locations acceptable to retrieve articles and events.</td>
</tr>
<tr>
<td>start_date</td>
<td>String (YYYY-MM-DD)</td>
<td>Yes</td>
<td>First day of the interested time frame for which we want to calculate media attention features.</td>
</tr>
<tr>
<td>end_date</td>
<td>String (YYYY-MM-DD)</td>
<td>Yes</td>
<td>Last day of the interested time frame for which we want to calculate media attention features.</td>
</tr>
<tr>
<td>event_feature_id</td>
<td>String</td>
<td>Yes</td>
<td>Identifier to filter features in the feature selection phase.</td>
</tr>
</tbody>
</table>

### 3.3.3.2 Media attention feature selection

To select media attention features during model building, we use feature_selector as described in D3.2. In addition to providing the features list, we also filter media attention features only related to...
the specific concept and region. This is possible by using event_feature_id and QMiner's query language32.

**Example**

How the API for media attention features can be used in practice is best seen with an example. Let’s say we want to build the model for a category "/Sport & Fitness/Sport/Fußball", retrieved using the clustering keyword tool, for any region inside Germany for the year 2017, we would specify media attention feature transformer as:

```json
{
    "module": "events_features",
    "params": {
        "download": true,
        "input_db": "../data/usecase/jot/common/dbs/jotEventsDb",
        "output_db": "../data/usecase/jot/common/features/jotEventsFeaturesDb",
        "clean_db": true,
        "queries": [
            {
                "keyword": "Association football",
                "locations": ["Germany"],
                "start_date": "2017-01-01",
                "end_date": "2017-12-30",
                "event_feature_id": "FootballGermany"
            },
            ...
        ]
    }
}
```

This calculates all media attention features within Germany for the Wikipedia concept "Association football", which is probably the best match for a category "/Sport & Fitness/Sport/Fußball". All features are also marked with ID "FootballGermany".

In a configuration JSON for the model building, we would define feature selector as:

```json
{
    "module": "feature_selector",
    "params": {
        "input_db": "../data/usecase/jot/common/features/jotEventsFeaturesDb",
        "forecast_offset": -1,
        "search_query": {
            "$from": "EventsFeatures",
            "EventFeatureId": "FootballGermany"
        },
        "features": [
            "EventsCounts",
            "ArticlesCounts",
            "MentionedArticlesCurrent",
            "MentionedEventsCurrent",
            "ArticlesCountsPastMax",
            "ArticlesCountsPastMin"
        ]
    }
}
```

"EventsCountsPastMax",
"EventsCountsPastMin",
"ArticlesCountsPastAvg",
"EventsCountsPastAvg",
"MentionedEventsFuture",
"MentionedArticlesFuture",
"MentionedArticlesFuture1",
"MentionedEventsFuture1"
}],
"normalize": "scale"
}

With the search query and the field "EventFeatureId", we can easily filter out media attention features calculated only for "Association football" inside Germany. In simple terms, with the search query, we select compelling rows from the database, and with the "features" list, we choose compelling columns. Field "forecast_offset" set to -1 dictates that the model predicts one day in advance. In other words, the pipeline fits the model on features from yesterday.

### 3.3.4 Relation to the Custom Events Model

To capture information about internal events in businesses a custom event ontology was developed. Its description and specification are presented in D1.4. Even though its target purpose is to hold information about business event it is general enough to model information about media events. This is beneficial as it offers the option to unify data processing pipelines and enables processing and storage in the same infrastructure.

The media event information is packaged into **media attention events**. Conceptually, these are not events that correspond to some real-world occurrence but package the signal about media activity into a computationally practical form. Each such event holds all the media attention information computed for a target date specified using schema:startDate and schema:endDate properties and a target region specified using the schema:location property. The media attention features are computed with respect to some topic. In the Event Registry data model this corresponds to a Concept or Category identified by its URI which can be encoded in the media attention event using the schema:category property. Finally, the feature values themselves can be included using the ews:quantity property and using the features’ names as their ews:quantityUnitId.

### 3.4 Key Updates Since the Previous Release

As covered in detail in this chapter, the key updates to the analytics toolset expanded its functionality in two directions. The first one, described in section 3.2, deals with clustering of keywords for processing meant primarily for the JOT business case. The other one, described in section 3.3, focuses on the enrichment with news event data obtained from the Event Registry.

---

[33] https://github.com/EventRegistry/event-registry-python/wiki/Data-models
Chapter 4  Data Visualization and Navigation

Data Visualisation and Navigation in EW-Shopp is offered mainly through the Knowage platform, customized for the user needs and scenarios carried out in the EW-Shopp pilots as well as other similar data visualization cases, in particular ones where business data (indicators) are coupled to weather/events data, and more typically along time series of varying time granularity.\(^\text{34}\)

Knowage is an open source business analytics suite developed by ENG and offered both as a Community Edition and with various Enterprise edition licenses. Among the different Knowage tools are a set of services able to visualize, navigate and explore in a rather intuitive way data present in most widespread data storage formats and systems. Being completely open source, Knowage is extensively documented online.\(^\text{35}\) In this chapter we focus on specific features and customizations used in EW-Shopp and in support to the EW-Shopp business cases as well as the updates compared to the previous release of the EW-Shopp Toolkit. Details about requirements of Business Cases (including those relating to Visualization) and how these are fulfilled by the EW-Shopp toolkit can be found in D2.3 and D2.4 EW-Shopp Platform evaluation assessment deliverables where more details about the visualization of each single business case are also provided. At the time of writing (December 2019), the reference Knowage version for EW-Shopp has been updated to version 7.0 of the Community Edition which is in line with the latest upstream version (released October 2019).\(^\text{36}\) This also means that currently in EW-Shopp the fully Open Source version of the suite is being used, possibly allowing anyone to further use it for free.

34 To this end please refer to D2.3 as well as D2.4 where a full use case of this type is presented


36 See https://www.knowage-suite.com/site/knowage-7.0-now-available/

Figure 10 Main Knowage website (left): www.knowage-suite.com – EW-Shopp Knowage 7.0 instance login page (right)
4.1 Data Visualization and Navigation Services

In this section the Knowage’s main features as well as its resource requirements are presented. The reader is referred to D2.4 for a more practical and weather-oriented set up.

4.1.1 Factsheet

<table>
<thead>
<tr>
<th>Knowage</th>
<th>Knowage is the open source suite for business analytics that combines traditional data and big data sources into valuable and meaningful information. Knowage supports a modern vision of the data analytics, providing new self-service capabilities that give autonomy to the end-user, now able to build his own analysis and explore his own data space, also combining data that come from different sources. Key features include:</th>
</tr>
</thead>
</table>
| Functionality | • Data federation and mash-up  
• Mixed datasets (types and sources)  
• Advanced data visualization  
• WYSIWYG visualisation dashboard creation  
• Server and user management including permissions |
| Usage in EW-Shopp | In EW-Shopp Knowage is used as the main Data Visualisation and navigation component |
| API documentation | [https://knowage.docs.apiary.io/](https://knowage.docs.apiary.io/) |
Knowage is released with a dual licence:

- Community Edition – License: GNU AGPL v3.0
- Enterprise Edition

**Source code repository**

https://github.com/KnowageLabs/Knowage-Server

**Contact person**

Lorenzo.sutton@eng.it

### 4.1.2 Data Acquisition and Management

At the heart of data visualizations services available in Knowage is the concept of *data source* and *data set*. A data source is any of the supported systems (such as a relational database) Knowage is able to connect to in order to extract and process data which will eventually be used for visualization. Figure 11 shows an example of MySQL data source for BC3.

Additionally, Knowage offers a **deeper level** of data-interaction ‘granularity’ through Data Sets. Within Knowage, Data Sets act as data providers for visualization and navigation documents and therefore support several types of data representations and provisions. Some of these include:

- Files (e.g. CSV)
- Queries (e.g. SQL query)
- REST calls
- Java Classes
- Scripts
- Ckan
- Federated

---

37 See: [https://knowage-suite.readthedocs.io/en/7.0/functionalities-guide/basic-data-access/index.html#dataset](https://knowage-suite.readthedocs.io/en/7.0/functionalities-guide/basic-data-access/index.html#dataset)
A very relevant feature of non-static Data Sets (in particular Queries or REST), is the possibility to **dynamically parametrize** certain values. For instance, an SQL query can contain parameters which will be set dynamically by user interaction in the visualization environment. In this case, if we consider a query data set (e.g. SQL), such query will be executed each time with the different values contained in the parameters and set, for example, by the user interacting with the visualization cockpit or through certain lists of values (see also below for a detailed description). Figure 13, Figure 14 and Figure 15 show various examples of data sets used for implementing the visualization part of the EW-Shopp Business Cases.

For all types of data set it is possible to **preview** (Figure 16) in real time the data set (e.g. the result columns and rows of a query), and additionally select types (e.g. number, date, string) of each field. Additionally, for each field of a data set **role metadata** in terms of **attribute** or **measure** should be defined: in fact, depending on their role, data can be used differently when explored and visualized (for example the X value of a typical histogram chart should be an attribute while the Y should be a **measure**). It should be noted that when creating a data set, Knowage tries to guess the role of a data field, but this can be easily customized by the user. Figure 17 shows an example of role metadata definition for BC3.
Figure 12 Accessing Data sets from the main Knowage 7.0 menu

Figure 13 CSV Data Set for Business Case 3 (Measurence)
Figure 14 SQL query Data Set for business case 3. Notice in the SQL snipplet shown, the dynamic parameters set with the syntax e.g. $P{storeID}$ and $P{day\_week\_1}$. 
Figure 15 A REST data set in Knowage, in this case the ArangoDB connector for JOT Business Case 4. Also in this example a dynamic parameter is used.

Figure 16 Data set preview for one of the BC3 data sets. If the query contains parameters the user is able to insert them before previewing or use the default values.
In order to manage, filter and present data, Knowage offers a set of behavioural models and analytical drivers. Here we highlight the possibility of defining Lists of Values (LOVs) from data sets which can then be linked to documents and essentially determine (still through user interaction) the way data is presented. For example, in the Measurence Business case the user typically wants to store the store to analyse (and a dedicated data set which retrieves all store data is created). Figure 18 shows an example of LOV with all available stores defined for BC3. The LOV can then be used in cockpits to dynamically drive the visualisation (see Figure 18).

38 For detailed documentation about Behavioural Models see: https://knowagesuite.readthedocs.io/en/latest/functionalities-guide/behavioural-model/
4.1.3 Visualization and Navigation Tools

Once the data sources and data sets to be used in the business case have been defined, the actual navigation and visualization is implemented through the creation of Cockpits. Cockpits allow to build interactive environments through WYSIWYG\(^\text{\ref{wysiwyg}}\)\(^\text{\ref{wysiwyg}}\) and intuitive interfaces (e.g. clicks and simple drag and drop). In this way analytical documents can be composed with multiple visualization and navigation widgets, easily defining associations and interactions: for instance, clicking on one widget, data in all other widgets is automatically updated. Figure 19 shows the widget selection dialog where users can visually select widgets to be added to a cockpit.

\(^{\text{\ref{wysiwyg}}} \text{What You See Is What You Get} \) \(\text{https://en.wikipedia.org/wiki/WYSIWYG}\)
Within the cockpit **editing environment** it is possible to add directly different Data Sets to be used for the data visualization and navigation and, most importantly, the user can visually create **associations** between different data sets.

**Associations** are a very powerful feature which enables to link fields (i.e. columns) of one data set with the field from a different one. Once associated any change to one field of a data set (e.g. user selection) will be reflected in real-time in all associated data sets, allowing for creating quickly **highly interactive** cockpits. Figure 21 shows an example of visual data association for BC3: users interactively associate fields (i.e. columns) from the different data sets.
Figure 21 Visually creating parameters (i.e. columns) associations between data sets in a cockpit.

Once the needed data sets have been added, various **visualization and data navigation widgets** can be added to the canvas of the cockpit. The set-up in terms of behaviour, data as well as visuals is controlled by a consistent and very intuitive user interface, additionally allowing (for more advanced widgets like charts), to manually tweak several parameters by more expert users. Figure 22, Figure 23 and Figure 24 show various widget editing screenshots.

Figure 22 Setting up a chart for the Business Case 3 weekly store comparison cockpit
All widgets are ‘data-aware’, including (for instance), selection lists (like dropdowns or combo-boxes), text and HTML widgets. **HTML** widgets in particular allow to use standard HTML while adding dynamically data from the data sets. Figure 25 shows an example of adding an OpenStreetMap to the Measurence store analysis with location of the store retrieved from the store information dataset (which contains coordinates and address), then embedded in a parametrized HTML widget. Notice the templating feature where parts of the HTML can be dynamically created based on data sets. A similar dynamic behaviour can be achieved, for instance, in text widgets where parts of the text can be extracted dynamically (and parametrically) from cockpit data sets (see Figure 26).
As explained above, behaviour and interaction can be enhanced and augmented through LOVs, for example in order to enable a user to select a specific item within a large database. In the figure below a store selection window (accessed directly from the cockpit) is shown as well as a more complex product search (through a LOV). In the former the selection is provided through a checkbox interface; in the latter the search is incremental (i.e. results are shown as the user types), allowing to quickly retrieve matches even in large data sets such as product catalogues or user directories. Query parameters can be saved for quick future retrieval and use as the figures below illustrates.
Figure 27 Two examples of parameter selection: left checkbox style (used in BC3). Right: searching within the BC11 product catalogue. Once selected the parameters will be used with the connected datasets.

Figure 28 Query parameters for LOVs used in the cockpit can be saved for quick retrieval.

Of course, parameters from the LOVs selected by users can also be used in data set associations and, in turn, used to personalized both widgets and queries, etc.
Figure 29 The selected parameter is used in most widgets using the same data set highlighted by the yellow boxes: in this example the title text, the chart and the HTML widget all rely on the parametrized selection of the store location (note some data blurred for confidentiality reasons).

As already anticipated above, cockpits can also be **customized** and edited in a more advance manner through the specific **Document Details** interface. From here the analytical drivers (including behavioural LOVs) can be managed, for instance added, renamed, etc. Figure 30 shows two example screenshots of the document editor where such advanced features can be edited.

Figure 30 Advanced document editing for the cockpit including (below) Analytical driver set-up
4.1.4 Server and Instance Management features

Knowage Open Source Community Edition (CE) can be freely downloaded and installed by anyone and is cross-platform running on Linux and Windows. The suite is distributed as a one-file installers and in other formats. The basic requirements are a machine with at least 3 GB of dedicated RAM and 2 GB of free disk space a Java SE Development Kit (JDK). For the EW-Shopp Business Case pilots a dedicated instance was set-up since April 2018, running on ENG’s managed cloud, and updated at different stages first to Knowage v 6.4 and then – most recently – to Knowage 7.0 CE.

Being a business-oriented cloud suite, Knowage has a wide set of server- and administration management features documented in detail. Here we will highlight some of the administration services particularly relevant within the EW-Shopp scenario. These are namely Users and Roles management. Users can be added and managed directly from the online interface by an administrator user.

![Image](image_url)

Figure 31 Knowage user management menu (above) and interface (below)

User capabilities and how they can interact with the platform in terms of data and interfaces is set in a rather granular way through the assignment of roles. This has the advantage that a predefined role

40 See https://www.knowage-suite.com/site/knowage-download/
41 This refers to version 6.2.1 which is the current release version at the time of writing.
with certain permissions and characteristics can be set-up (i.e. like a template), and then assigned to multiple users. Additionally, a user can have multiple roles.

Among the many authorizations and permissions’ granting applied through roles, it is relevant to mention here: capability to manage (or not) other users, permission to create new documents (vs. only viewing them), and available data sets. This latter feature is particularly important because it enables to control which data sets a user can see (or not). Therefore, enabling different users to only see and use relevant data sets as well as (for example) to hide certain data sets from certain users: for example, in the case of a data analyst and final users both using the platform the analyst might be granted use and visualization of certain data sets while the final user only of a restricted set due to privacy/confidentiality measures. Figure 31 and Figure 32 show screenshots of the user and role management interfaces.

![Figure 32 Setting up a typical EW-Shopp business case user role in the administration interface](image)

Finally, most server settings for the Knowage instance in use can be directly accessed and modified by an administrative user directly from the **Server Settings** section of the online interface. The interface also includes incremental search for the configuration parameters. Only users with server administration privileges are exposed the server configuration management area.

![Figure 33 Server configuration interface, including incremental search of parameters](image)
4.1.5 Visualisation as a Service (VaaS) through Knowage API

In addition to front end user (web) interfaces, Knowage allows to access most parts through a REST API, essentially allowing Visualisation as a Service. The idea is that most components set-up in Knowage (for instance data sets, cockpits, etc.), can be ‘consumed’ externally (e.g. on a clients’ website or legacy system), through an API. The Knowage API is extensively documented and maintained online at Apiary. Additionally Knowage documents (and cockpits) can be embedded in external HTML (provided the right authorisations are in place for the user). In the figure below a logical workflow (based on the EW-Shopp dataflow methodology) for Business Case 3 (as had been presented in D2.3) is augmented with a potential API/embedding step to provide, for instance, some cockpits to clients in their own environment.

Below are example screenshots of potential interaction with Knowage as a Service taken from BC1 (API access to a data set) and embedding related to BC3, where Measurence, as well as accessing the online Knowage instance, could be interested to query Knowage data sets as well as integrate some cockpits in their internal or clients’ interfaces.

43 See https://knowage.docs.apiary.io/
Figure 34 Retrieving a Knowage data set via REST API in Postman (above) and with Python requests (below)
4.2 Key Updates Since the Previous Release

The first most evident update is the adoption of the Knowage Community Edition (vs. the business edition previously tested in EW-Shopp), allowing easier sustainability and replicability, and upgrade to v7.0 (after an intermediate update to 6.4 - for a detailed feature lists see here: https://www.knowage-suite.com/site/knowage-6.4-release/ and http://release.ow2.org/knowage/7.0.0/Knowage-7.0-release-notes.pdf).

As seen in the screenshots above the interface is completely updated with more evident and minor updates (e.g. colours in data set preview columns). The behavioural model is now managed in the cockpit advanced editing interface (also completely upgraded) which makes the use of LOVs easier. Various widgets, including the HTML one, have been updated. More specifically to EW-Shopp Knowage has been further and fully tested with Business Case 3 (Measurence) allowing to show various features such as functionalities management, HTML widgets with iFrame, different kinds of updated charts (e.g. with multiple categories); additionally a general weather-data based use case has been developed with this version (described in more detail in D2.4), and allowing to demonstrate the re-usability of the tool.
Chapter 5  Other Supporting Tools and Services

This chapter is devoted to present other supporting tools and miscellaneous work not attributable in particular to any one of the main services comprising the toolkit but relevant to cover unforeseen needs emerged during the project. In particular, in this chapter are presented:

- the deeplearning-based Product Matching Algorithms developed by UNIMIB and Ceneje (Section 5.1);
- the service based on the semantic publication of GfK’s products catalog, named CatOD - Catalog on Demand (Section 5.2);
- a research tool named MantisTable, developed by UNIMIB as testbench for advanced annotation algorithms (Section 5.3).

5.1 Product Matching Algorithms

UNIMIB has worked in cooperation with Ceneje to devise Product Matching Algorithms that can improve the capability of Ceneje – and similar companies, e.g., their sister companies covering markets for different countries and languages in the Heureka Group – to match different product descriptions and consolidate the data space used by comparison shopping platforms (CSPs).

Before presenting the work carried out, we briefly introduce some terminology. In CSPs, different merchants advertise different product offers, also referred as offers for simplicity. As a result, the same product may be available in different product offers as provided by different merchants: the CSP uses mappings between different offers corresponding to a same product and this product to group offers when displaying results for users’ searches. The capability of grouping several offers corresponding to the same product is at the very heart of a CSP and crucial for providing users with high-quality data. Through the integration of offers that correspond to the same product, a CSP as Ceneje builds its product catalog (some CSP owners buy product catalogs from third-party sources, e.g., the GfK Catalog also described in Section 5.2).

Matching product data is a flagship problem for matching algorithms because it is rather challenging. Usually, the offer titles contain a reference to the product brand and model. However, different merchants may use different attributes to describe the products, may use words in different orders, add catchy words to appeal the users, and even try to cheat the matching algorithms to gain more views (for example, cheating the algorithm to pass a phone cover for a phone itself may lead to better ranking by price and thus more traffic on the merchant website). Also, the offers may use words that refer to popular brands, but also to unknown ones, frequently used words in a language, e.g., “phone”, but also codes that mix letters and numbers and are not part of any natural language. Finally, matching methods are often sensitive to a language.
CSP have often developed techniques that are effective for the cases they know how to handle but more brittle when infrequent word patterns are used to describe the offers. Typically, preprocessing is an important step and algorithms try to match codes, which acts as unique identifiers for products. However, these codes may not appear in many offers, or may be written by splitting components of the code, or by merging the code with some other characters. As a result, code-driven matching algorithms often achieve very high precision, but may encounter recall problems, failing to find matching offers when codes are not written in the expected way or are absent. This was the case we found for Ceneje product matching algorithms, on some product categories. Ceneje algorithms, based on a long-term expertise on the domain, are very precise when codes exist and they can accurately find a match for several product categories, but fail to find matches for some categories and several product offers. Finally, product matching in real-world CSP also uses human-in-the-loop. For example, Ceneje has few employees to review matching results from their algorithms and select the correct matching product from a list of the three most similar products returned by the algorithm.

UNIMIB contributions has been targeted to mitigate this problem, searching for matching algorithms that:

- can be combined with precise code-driven matching algorithms of Ceneje to improve the recall, and, therefore, the overall number of matching found;
- can be used in the same way as Ceneje’s algorithms are used, i.e., also in interactive matching with experts validating the results of the algorithm;
- can be extended to different languages to be ready for future exploitation by the Heureka group.

We decided to bring to Ceneje some of the very latest research results in matching technology, which has also been applied to product data. We selected DeepMatcher⁴⁴, an open-source matching algorithm licensed under the 3-Clause BSD License⁴⁵. The tool implements deep-learning techniques to support matching, which provide a powerful but modular and principled approach to matching. Given a pair of entity descriptions, each one represented by values for comparable attributes, DeepMatcher assigns to the pair a matching degree, which, with a threshold, can be commuted into a class prediction (match vs. nonmatch). The algorithm process CSV files with 2k columns, where each product is described by values of k attributes.

---


⁴⁵ [https://opensource.org/licenses/BSD-3-Clause](https://opensource.org/licenses/BSD-3-Clause)
5.1.1 DeepMatcher: Short Overview

The tool contains several configurable modules orchestrated in a deep neural network in such a way that a classifier at the end of the network determines a matching score – through a classifier – for an input entity pair. The information flow is summarized in Figure 36. The tool obtains an embedded representation of the values of each attribute using word embeddings: each token that constitutes an attribute value, e.g., “nikon coolpix w100 fotoaparat moder nahrbtnik” (the value of the attribute “title” of product offer of Ceneje), is associated with a word embedding. Pre-trained FastText word embeddings\(^{46}\) in multiple languages can be used. The list of embeddings associated with each value can be summarized using different methods (e.g., average, attention-based summarization, etc.). Pairs of summarized attribute values are fed into nodes that determine the attribute-wise similarity score for the entity pair. These similarity scores are further combined to determine a final matching score through the classifier.

![Diagram of DeepMatcher architecture](image)

Each component can be configured in multiple ways. An overview of the configuration space is provided in Figure 37. Each layer in the neural network architecture can be trained again or use pre-trained configuration. Obviously, an effective use of the algorithm should at least train the latest

layers, while word embeddings can be used as available in pre-trained models. Several trade-offs between accuracy and efficiency have been discussed in the paper, so that the architecture can be configured to work with different resources.

<table>
<thead>
<tr>
<th>Architecture module</th>
<th>Granularity:</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute embedding</td>
<td>(1) Word-based</td>
<td>(3) Pre-trained</td>
</tr>
<tr>
<td>Attribute similarity representation</td>
<td>(2) Character-based</td>
<td>(4) Learned</td>
</tr>
<tr>
<td>Attribute summarization</td>
<td>(1) Attribute summarization</td>
<td>(1) Heuristic-based</td>
</tr>
<tr>
<td></td>
<td>(2) Attribute comparison</td>
<td>(2) RNN-based</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3) Attention-based</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4) Hybrid</td>
</tr>
<tr>
<td>Classifier</td>
<td></td>
<td>NN (multi-layer perceptron)</td>
</tr>
</tbody>
</table>

Figure 37. DeepMatcher configuration space – from SIGMOD18 paper.

From the above description it is clear that the algorithm fits one input requirements, i.e., the possibility to be adapted with limited effort to different languages so as to be exploited by Ceneje on the Croatian language and by other companies in the Heureka group. Also, the algorithm can be combined with precise code-driven matching algorithms of Ceneje with a simple and safe approach, i.e., by using it for offers that do not get any match from Ceneje’s algorithm. Instead, the algorithm requires adaptation to be used in the same way as Ceneje’s algorithms are used, i.e., also in interactive matching with experts validating the results of the algorithm. In fact, the algorithm is thought for deciding which entity pair matches or does not match, while in Ceneje human-in-the-loop framework, the matching algorithm must return a list of candidate products for each offer. Finally, DeepMatcher is as powerful as complex, meaning that the configuration space is really huge, from the selection of attributes to be considered, to the configuration of each module. Training in DeepMatcher requires the creation of nonmatching pairs, which is subject of domain-specific adaptation. Also, we observe that being Slovene a low-resource language, the effectiveness of pre-trained embeddings needs to be tested.

### 5.1.2 UNIMIB’s Product Matching Service

The main aim of UNIMIB has been to experiment DeepMatcher with Ceneje’s data and encapsulate the tool into a service that can be used in the Ceneje matching framework. The service consists therefore in a codebase, more than in a full-fledged service, due to the complexity of the solution and the need for further optimization. We can summarize the tasks carried out by UNIMIB and the resulting components of the code developed as follows:

- We have created a fork of the DeepMatcher codebase with significant changes. Changes have been directed to: make the tool compatible with latest PyTorch releases and therefore with the use of GPUs, which are widely used in deep learning to make training much more
efficient and are available at a limited (sustainable) cost; fix the code in several points to make it work smoothly.

- We have written several **scripts** to use DeepMatcher with Ceneje data and test its results, by covering in particular:
  - Preprocessing of the text (lower case, stop words removal, etc.)
  - Implementation of baseline matching algorithms (e.g., Jaccard similarity, edit distance, combination of baseline similarity measures using logistic regression with training data), which can be used for comparison, but, more importantly, for training as further explained below
  - Creation of DeepMatcher-compliant data to train, validate and test the algorithm from Ceneje’s input (including methods to create negative samples using different methods – see below)
  - Implementation of the script for **runtime execution of DeepMatcher** with creation of the data in the format required by DeepMatcher and ranking of matching descriptions to fit the algorithm to Ceneje workflow.

The code released by UNIMIB is available on GitHub\(^{47}\). The description of the approach, the experiments and several details are discussed in a presentation used throughout the work to share information with the Ceneje team. The presentation is available on GitHub\(^{48}\). Here below we briefly report some key elements of the work done and refer to the presentation for more details.

### 5.1.3 Running DeepMatcher on Ceneje’s data

Differently from traditional matching algorithms, deep learning algorithm requires two different environments:

- An environment for training the algorithm and testing its performance before applying it to unseen data, which usually considers an input **gold standard** that needs to be split into training, test, and validation data.
- An environment for using the algorithm on new data.

This is consistent with the application of DeepMatcher to product offers used by Ceneje, where manually validated mappings are available at a given point in time for a category of products (the cold start problem for a new category was not addressed in the current work) and we want to learn from the gold standard how to match unseen new offers, i.e., offers of new products that are frequently introduced in the market for a new category.

---

\(^{47}\) [GitHub](https://github.com/UNIMIBInside/Prodmatch)

\(^{48}\) [GitHub](https://github.com/UNIMIBInside/Prodmatch/blob/master/Ceneje_product_matching.pdf)
UNIMIB and Ceneje teams have collaborated was to identify which data to use for training and testing given a gold standard (already matched products) for a given category. We have also identified attributes to consider. Since the goal was to target recall more than precision, we focused on attributes that are available for most of offers (DeepMatcher is robust against missing values, although missing values can affect its performance). The identified attributes are:

- Title of the offer, e.g., “nikon coolpix w100 fotoaparat moder nahrbtnik” is the title of an offer for the Camera category
- Brand of the product advertised in the offer, e.g., “Nikon” is the brand of the previous offer.
- Description of the offer, a longer text that describe the offer by describing additional features in an unstructured format.

After testing, we decided to focus on title and brand, because the description is often too noisy to be helpful.

DeepMatcher requires a CSV that contains pairs of matching / nonmatching products for training and testing. So, we have agreed with Ceneje on a format to get data from their CSP and transform them in a way that 1) the CSV file can be used in DeepMatcher and 2) we can create data for training and testing of the algorithm. The script takes as input three files from Ceneje:

- A CSV containing the representation of a set of offers with k attributes (Title, Brand, and Description)
- A CSV containing the representation of a set of catalog products with k attributes (Title, Brand, and Description)
- A CSV containing the mapping between the offers and the product they match to; the mapping is assumed to be complete (two offers match if and only if they are mapped to the same product)

The algorithm is trained over matching and nonmatching offers; in fact, product representations are usually offers that have been lifted to be representative of a product in the Ceneje catalog. Reference product representations are built by matching offers. Once a product representation is created it also has a Title, a Brand and a Description, plus other attributes not always available for offers. In other words, by learning from offers pairs, we cover the information that it is available when trying to match offers with existing product representation as well as the case where we want to match offers against other offers. Figure 38 shows an example of training data created from Ceneje’s offers using Title and Brand as attributes. The label on the right-most column specifies which are positive and negative examples.
Starting from a set of offers and/or products that are known to match from the Ceneje mapping, we set up a training, testing and evaluation environment. In this setup, we had to consider one aspect relevant for both training and testing. Obviously, we can have several offers that map to the same product. However, when applying the algorithm in production, it can be the case that only one offer match to a given product. So, we wanted to make sure that the algorithm does not overfit on repeating offers and generalize across the product. Also, we wanted to make sure that easy-to-match offers that are frequent in the gold standard, do not skew the evaluation results. We therefore tried to run training and evaluation in a pessimistic mode, i.e., by removing duplicates. We say that a dataset does not contain duplicates if and only if all the offers in a dataset correspond to different products.

The process to create data to support training, testing and validation are defined as follows:

- 15% of offers removing duplicates are put aside as unlabeled test data; this data is set aside to evaluate the performance of the algorithm when executed in ranking mode, i.e., mode compliant with Ceneje matching framework. In ranking mode, we want to order matching products for an input offer. To test this in production workflow, we want to use offers of products that have not been seen by the algorithm during training. This will make it more difficult the job for the algorithm but will provide worst-case scenario results on the performance (the case where we are matching offers for products never seen during training). Also, we can use this dataset to compare DeepMatcher results with unsupervised algorithms.

- Out of the remaining 85% of offers, we create three datasets: training set, validation set and test set. The procedure is introduced because we want to test different algorithms (including different DeepMatcher configurations) against one agreed test set – in addition to the set of unlabeled test data -. and there is no possibility to create negative or positive examples at runtime. So, it works as follows;

  1. We first create one dataset consisting of positive and negative samples, referred to as global positive and negative sample set, as follows:
1. We create all the unique pairs of matching offers using available mappings (offers match if they are known to match with the same product). This is considered the global set of positive samples, i.e., positive matches.

2. For each positive matching pair, we create $n$ (with $n=3$) negative samples, i.e., nonmatching pairs according to the gold standard. This is created taking care of similarity between nonmatching pairs as follows:
   - (The ceiling of) 50% of the $n$ negative samples created for each product is created by finding the most similar products to the source product (computed as Jaccard similarity on the name attribute); the remaining 50% of negative samples is created randomly from products within the same category.

2. Once we have this dataset, we create the test set; we take 20% of the global positive and negative sample set (using stratified sampling, which preserves the proportion of $1/n$ between positive and negative samples) in such a way that we have at most one positive sample for each offer (i.e., offers in the test set appear without duplicates). Observe, that this test set contains positive and negative samples, so it is used to test the algorithm in classifying offer pairs as match vs nonmatch (the evaluation settings traditionally used in academic evaluation and also reported in the original SIGMOD18 paper)

3. Then we extract all the offers that are different from the ones that appear in the unlabeled set and in the test set. For all these offers, we repeat the process described for the global positive and negative sample set and obtain a dataset with positive and negative samples. Then this dataset is evenly split in training set and validation set.

The proposed data creation approach looks a bit complex, but it serves two important purposes 1) make sure that in the test set we do not have offer pairs that are equal to the pairs that have been used in unlabeled set and in the training + validation sets; 2) make sure that we can modify the approach used to train DeepMatcher, keeping a reference and shared test set. In conclusion, for each category we will have two sets for validation that can be used to compare different algorithms:

- A test set, which consists of matching vs nonmatching offer pairs, where performance is evaluated in the classification task of deciding whether a pair belong to the class matching vs nonmatching; this test set contains offers that have not been used during training. This can be used to evaluate the performance of different configurations of DeepMatcher but cannot be used to compare DeepMatcher with Ceneje algorithms, which do not work as classifiers.

- An unlabeled test data set, which consists of offers and their matching products, where performance is evaluated in ranking mode, i.e., by looking at the ranked list of products returned for each offer. Also, this set contains only offers that have not been used during training. This can be used to evaluate the performance of different configurations of
DeepMatcher and to compare DeepMatcher with Ceneje algorithms, which natively work in ranking mode.

5.1.4 Tested Configurations of DeepMatcher

We conducted several experiments to test different configuration of DeepMatcher. We searched in the configuration space that does not require too many resources to train. We first conducted experiments with test sets and then with unlabeled test data sets to evaluate the performance with Ceneje algorithms. The configurations tested are summarized here below:

- Attributes considered by the matcher, one or more of the following attributes: Brand, Title, Description).

- Different training strategies:
  - Selection of positive examples:
    - titles as they appear in the offers (after preprocessing);
    - 50% of the offers have titles as they appear in the offers and 50% of the offers where tokens appearing in the title are recombined randomly (order of tokens is scrambled).
  - Creation of negative samples (proportion with respect to positive samples is always 1/3)
    - Random (selected from other products in the same category that are known to be non-matches for the given product)
    - Heuristic selection of negative samples:
      - (Ceiling of) 50% of negative samples is taken from the most similar products to the offer based on Jaccard similarity over the tokens in the offer title
      - 50% of negative samples is selected randomly

- DeepMatcher configuration:
  - Word embeddings: we always used FastText pre-trained embeddings on Slovenian language
  - Attribute summarization:
    - RNN - YES (with GRU)
    - Attention - NO (tried in early experiments and discarded because not promising)
    - Hybrid - NO
  - Other parameters are left unchanged from suggested default configuration
5.1.5 Conclusions

Results of the experiments are reported in the presentation that we link to the code base, which we continue to update with new results. We summarize the conclusion of the experiments as follows:

- DeepMatcher is a powerful, highly configurable, and multi-lingual tool; it is particularly useful to match offers using a limited number of attributes, thus working with a large number of categories whose products are described with little information; pre-trained embeddings make it easy to port the algorithm to markets with offers described in different languages.

- DeepMatcher works well also with information coded in a limited attribute set; we found that using Title and Brand achieved better performance that including also the Description. We could not try to use numeric attributes like price, which would require a substantial change to the algorithm and is left for future work.

- DeepMatcher has a large configuration space; finding optimal configurations is not trivial, with different configurations performing differently on different types of Titles.

- DeepMatcher and precision-oriented algorithms like the one by Ceneje, resulting from deep domain expertise, are more complementary other than in a direct completion. We found that the soft-matching approach implemented by DeepMatcher is not as accurate as Ceneje’s algorithm on offers where Ceneje smoothly leverages the codes that identify product models. We do not believe that further optimization can beat those precision-oriented algorithms. On the other side, DeepMatcher is able to find matches for offers that do not get any result from Ceneje’s algorithm. This is also due to the fact that Ceneje’s algorithm produces either reliable results (with correct product ranked in the three top positions) or no result. The cornerstone issue is the usage of codes for matching: they are wisely used by Ceneje and not so well handled by DeepMatcher when using FastText embeddings (also because these embeddings are character based). However, discriminating between tokens (or sequences of tokens) referring to codes and tokens (or sequences of tokens) not referring to codes is not as trivial as it may sound. The solution we currently devised is to combine the two algorithms with a trust-based approach: if Ceneje’s algorithm finds results for an offer, these results are trusted; otherwise, the offer is processed by DeepMatcher. Obviously more advanced combination approach can be tried in future work.

- DeepMatch seems to have good potential for innovating product matching in Ceneje and Heureka group, and, in particular, to find matching offers when codes are difficult to be leveraged by precision-oriented algorithms. However, a thorough and systematic strategy to evaluate the performance of the algorithm for products of each category is needed. Such a strategy was developed within EW-Shopp, in addition to the code to implement this strategy on Ceneje’s data.

5.2 GfK CatOD (Catalog on Demand)
For reasons that have been explained in detail elsewhere, GFK was affected by a change in the organization that has prevented the company to work on a full-fledged business case as originally planned. As agreed with the European Commission and the Consortium Partners, and also formalized in an amendment to the GA, GFK (GFK Italy) has completed a technical task, that is, the implementation and deploy of a solution, also referred to as a (technical) service, to publish and make accessible their Product Catalog using semantic Web standards (RDF) or, in other words, as Linked Data, or, yet under a different perspective, as a Product Knowledge Graph. The solution is particularly relevant today as Schema.org is gaining momentum as a vocabulary for data sharing across the web, and thanks to more pragmatic semantic exchange formats such as JSON-LD. In addition, several companies, in particular product data aggregators like Amazon and eBay, are today developing solutions to transform their data into knowledge graphs and use semantic technologies and principles to support the integration of different sources. GFK catalog is one of the richest and most complete catalog of Consumer Electronic goods available as of today, thus making the GFK service very relevant in the today web data landscape.

The solution consists in an ontology to represent GFK product data, which is based and extends Schema.org (see D1.2 for more details about the ontology) to cover the requirements coming from GFK. In addition, the solution is based on the ingestion of tabular data formats and their virtual semantic publication using the D2R server, which makes it easier to update the semantic representation upon changes in the catalog. Thus, the solution includes the specification of the mappings that guarantee the publication, but also a software architecture that supports the publication of the data and their access using well-known D2R functionalities (SPARQL queries and web browsing), as explained in detail in D1.2.

Since we cannot consider this as a full-fledged business service (like the others described in D4.3) and a full business analysis of the service was not planned, we do not present its exploitation strategy with the details and structure we used for the other business services or components of the EW-Shopp toolkit, but adding a summary of the exploitation strategy as we did for other technical contributions (e.g., ABSTAT in UNIMIB’s individual exploitation plan). Despite this remark, GFK intend to exploit this technical contribution after the end of the project, as briefly summarized here below.

The general aim of the new solution is to support all the e-retailers, particularly the ones with a medium-low business size, to have access to a complete and quickly updated set of product features, in order to properly describe their product e-shelf, being more competitive in the current crowded and rapidly changing market scenario.

The need of a different approach comes from the specific circumstances of these Retailers: typically they don’t have the economic resources and the minimum required budget to access a “full Catalog” (e.g. GfK product Catalog) covering all the SKUs sold in the market, condition that generates a competitive gap against the bigger online Retailers; on the other hand, they often don’t have the necessary organizational structure to build, by themselves, proper product sheets, harmonized among different Manufacturers and complete in terms of product description details.

---

49 Stock Keeping Units
These limits could, in many cases, negatively affect the capability of these websites to generate online shoppers traffic as well as the ability to convert online visits in sales; the final result is a negative performance compared to the real potentialities they could have in the online channel, that continues to shows double digit positive trends, in Europe, for Tech & Durables.

To realize this alternative business approach GfK and UNIMIB worked on the solution design starting from the GfK Catalog, existing solution offered to the market players, to integrate the possibility to focus the selection, through new queries, to the real offer the online Retailer has to cover, allowing them to save money compared to a full access.

This new service design represents an evolution of the “core” work of EW-Shopp and, for this reason, it has not been included in a dedicated deliverable. Anyway, this alternative approach could be an important upgrade in the current GfK Product Catalog offer, as well as a step forward on a more client centric approach.

The dedicated web platform we are intended to develop will allow to extract only the required contents from the GfK Catalog Database in an easier and smarter way, covering the needs of an higher audience of E-Retailers, including the smaller ones.

5.2.1 From the Idea to the Prototype

This new approach, based on the effort of GfK and UNIMIB, has the main objective to allow all these players (that often are working today with marketplace platforms) to enrich their basket of services offered, thanks to structured information about all the main technical features, reducing the selection to the current need and saving money compared to a full access.

To realize this alternative business approach GfK and UNIMIB worked on the solution design starting from the GfK Catalog, existing solution offered to the market players, to integrate the possibility to focus the selection, through new queries, to the real offer the online retailer has to cover.

A workflow consisting of three main steps was therefore defined and implemented. A first fundamental step for this development has been to adapt the current GfK Catalog to be ready to be converted in an RDF (Resource Description Framework) semantic web language. To achieve this objective the main elements of GfK Catalog (e.g. “Brand”, “Product Group”, “Feature”) have been renamed and mapped onto concepts provided by several shared web vocabularies (e.g. GS1, Schema.org etc, see Figure 38).

```xml
@prefix map: <#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwi.ub-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
@prefix ewo: <http://product.data.ew-shopp/ontology/> .
@prefix gsl: <https://gsl.org/voc/> .
@prefix sdo: <http://schema.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```
This passage was meant to create an unambiguous description of all product features: for example what is currently coded as “EAN” code in GfK Catalog has been renamed in “sdo:gtin13”. Further, to cover all those cases in which the concepts represented in the GfK Catalog Database were not already coded in common shared vocabularies, a new ontology has been created from scratch (identified as “product.data.ew-shopp/ontology” in Figure 38).

In the second phase the product catalogue (contained in a relational datastore) was mapped to its semantic representation. The design choice in fact is to use D2R server50 to publish the data in RDF format. D2R exposes a Sparql51 endpoint keeping the data storage in a non-semantic datastore. Sparql queries are automatically and transparently converted to SQL and then run on the underlying database, eliminating the need to replicate the data in a dedicated RDF triple store. The final step has required the deployment of D2R and the related mappings on GfK’s premises, to implement a SPARQL endpoint to be exposed on a specific GfK web page.

Finally, it has been created a back-end service to extract and present, responding to requests of product information about a specific basket of SKUs, the corresponding product sheets. Customized queries have been set up behind the scenes to enable the web interface to interact with the RDF representation of the GfK Catalog.

The following screenshots showcase the current state of this first prototype and how it works. Figure 40 presents the D2R Server homepage. The user is given the option to explore it - either with a standard HTML view - using the links at the top or an RDF view - or, alternatively, query it using a SPARQL Explorer.

---

50 http://d2rq.org/d2r-server
51 https://www.w3.org/TR/rdf-sparql-query/
Figure 40 D2R Server homepage of GfK Product Catalog

Figure 41 presents an HTML based view of content (in its current state, the server is configured to display up to 50 entities at a time). In this case, the Product Categories are showcased. These categories are taken from the GfK Catalog, which is standardized in order - for the data - to be uniform across manufacturers and to allow the filtering and comparison of products.
To get product information from the catalog database, the above mentioned SPARQL Explorer can be used. Its interface is displayed in Figure 42.

![SPARQL interface](image)

**Figure 42 SPARQL interface**

Figure 43 shows a basic example of a query, in this case using the EAN code as input (which will likely be among the most common queries performed by customers). The product that matches the query is displayed at the bottom as a result, as a URI to the product data sheet.

![SPARQL query example](image)

**Figure 43 SPARQL query example**

The URI redirects to the data sheet displayed in Figure 44, with each additional URI corresponding to brand, features (*hasFeatureData*), etc.
Other examples of queries may include a more generic input, such as Brand name, Product category (see Figure 45), etc.
access to our platform. Thus, being able to request and receive, on demand, technical product data sheets that can be imported, implemented and published on client websites.

### 5.3 Mantistable

This section is meant to introduce MantisTable\(^{52}\), a Semantic Table Interpretation tool developed by UNIMIB, which allow the user to automatically annotate tables using a reference Knowledge Graph.

MantisTable and ASIA are closely related; in particular, MantisTable (which was recently awarded at the ISWC19 conference) represents the bedrock of cutting-edge approaches. Obviously not all the experimentations carried out within MantisTable have been implemented in ASIA, especially because the focus of the two tools is different. ASIA in fact is more oriented to the processing of large data sets and offers the functionality of creating knowledge graphs in RDF. In addition, ASIA offers the user enrichment functionality in addition to annotation (schema-level and instance-level). That being said, it seemed important to include a brief description of MantisTable in this deliverable because the great flow of ideas and insights experienced in EW-Shopp has brought great benefit to MantisTable and, vice versa, the experiments carried out on this tool have contributed, with a continuous inversion of roles, to the definition of techniques and algorithms implemented in ASIA.

MantisTable, like ASIA, provides a graphical interface allowing users to analyse the results of the semantic table interpretation process and validate the final annotations. The tool also provides a guided mode for viewing and editing annotations by users.

### 5.3.1 Automantic Semantic Table Interpretation

The MantisTable tool realises the semantic table interpretation through five phases:

1. **Data Preparation** aims to clean and uniform data inside the table. Transformations applied to tables are as follows: remove HTML tags and stop words, turn text into lowercase, solve acronyms and abbreviations, and normalise measurements units. The latter is performed by applying regular expressions.

2. **Column Analysis**, whose tasks are the semantic classification that assigns types to columns that are named entity or literal column, and the detection of the subject column. The first step of Column Analysis phase is to identify good Literal column candidates. The second step deals with the subject column detection that takes into account the identified named entity columns. We can define the subject column as the main column of the table based on different statistic features.

3. **Concept and Datatype Annotation** deals with mappings between columns headers and semantic elements in a knowledge graph. In the first step of Concept Annotation, the tool

---

\(^{52}\) [http://mantistable.disco.unimib.it/](http://mantistable.disco.unimib.it/)
performs the entity-linking by searching the knowledge graph with the content of a cell, to get a set of candidate entities. We use the DICE similarity measure between the content of the cell and the candidate entities to disambiguate the content of the cell. In the second step of Concept Annotation, the abstract and all concepts for each winning entity are retrieved from DBpedia.

4. **Predicate Annotation**, whose main task is to find relations (predicates) between the main column and the other columns to set the overall meaning of the table. MantisTable considers the winning concept of the subject column as the subject of the relationship, and the remaining columns as objects. In order to identify the correct predicate, the tool compares the content of the column and the candidate predicates.

5. **Entity Linking** deals with mappings between the content of cells and entities in the knowledge graph. The annotations are used to create a query for the disambiguation of the cell contents. If more than one entity is returned for a cell, the one with a smaller edit distance is taken.

### 5.3.2 The user interface

MantisTable is a web application developed with NodeJs\(^5\). The source code is released through a Git repository\(^6\). MantisTable supports the user to execute semantic table interpretation by means of a rich GUI, whose main elements are presented in Figure 46.

After selecting a table, it is possible to manage the execution of the five phases described in Section 5.3.1. The user can either run all steps together or run them step-by-step to supervise the execution (Figure 46 - info mode).

---

\(^5\)https://nodejs.org/
\(^6\)https://bitbucket.org/disconimib/mantistable-tool/
It is possible to navigate all the executed steps by clicking on each phase and analyse the results in the visualization mode (Figure 46 - 1. left side bar). For all phases, additional information about the execution is shown in the console located under the table (Figure 46 - 2. console). By clicking on a header or body cell, information about the current phase is reported in the info mode (Figure 46 - 3. right sidebar).

Even if MantisTable implements a fully automated annotation process, it is important to allow users to understand what has been achieved and give them the opportunity to modify and enhance the results. The former has been achieved with the exploration features sketched above, the latter has been accomplished by providing a widget to edit the annotations (Figure 46 - 4. edit mode). The annotation validation and editing require that the user has previous knowledge about the structure of the KG. Therefore, to support the user we integrate ABSTAT.
Chapter 6 Conclusions and Outlook

In this report we presented the EW-Shopp components which make the toolkit released by the project. EW-Shopp aims to support companies in providing a better ‘consumer journey’ and to leverage on the large amounts of data generated in such activity. Starting from a consistent Dataflow methodology agreed within EW-Shopp, services for reconciliation and enrichment of this data are provided enabling to utilize both internal or external data sources such as weather or geospatial data and, subsequently, enabling to carry out advanced analytics on it and eventually presenting them in a visually appealing and useful ways.

In presenting the final version of the complete EW-Shopp toolkit, we are aware (also thanks to interactions with business partners and end users) that not every partner necessarily uses every single tool from the toolkit. Therefore, we allow, and support through a fully modular and Open Source approach, the possibility to adapt data pipelines to fit users’ business needs.

Partners SINTEF and UNIMIB have built an infrastructure that can process large amounts of data. Processing means ingesting data into the infrastructure, cleansing it and enriching it. They tackled the big data problem by designing the Scalable data preparation and enrichment back-end in a way that enables scalability and parallel processing. So, alongside the data wrangling and enrichment tools (DataGraft, Grafterizer 2.0 and ASIA), an additional set of tools was utilized to enable scalability and parallel processing – Docker containers, GlusterFS and Rancher as the container orchestrator.

The enriched data set is then ready to be used by the machine learning algorithms of the QMiner component. All the business partners needed predictions based on historic data and JSI provided data analytics tools to fulfil this objective. Importantly, the provided implementations are fast, and the resulting data is propagated timely to business partners.

Finally, all the descriptive and predictive analytics needs to be presented in a visually appealing manner and the Knowage Open Source tool, developed by ENG and customized to the EW-Shopp scenarios, handles these aspects. Enabling various types of personalized queries and interactive visualisations such as charts, tables and HTML, and supporting many data connectors it enables both specialists and users to design and use interactive cockpits that are suited for the end user who can gain important insights from them.

Overall, we can conclude that of the components were successfully customised and improved following the requirements deriving from project activities, in particular in liaison with WP4 business cases. The fact that the EW-Shopp consortium adopted a strong Open Source philosophy since the beginning allows the tools to be available for anyone else willing to adopt them, use them or further develop them.
Appendix A  Data Enrichment Evaluation

Data enrichment refers to the process of appending or otherwise enhancing collected data with relevant data obtained from additional sources.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>#im</th>
<th>Region</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>194906</td>
<td>64</td>
<td>Thuringia</td>
<td>11/03/2017</td>
</tr>
<tr>
<td>517827</td>
<td>50</td>
<td>Bavaria</td>
<td>12/03/2017</td>
</tr>
<tr>
<td>459143</td>
<td>42</td>
<td>Berlin</td>
<td>12/03/2017</td>
</tr>
<tr>
<td>891139</td>
<td>36</td>
<td>Bavaria</td>
<td>11/03/2017</td>
</tr>
<tr>
<td>459143</td>
<td>30</td>
<td>Bavaria</td>
<td>10/03/2017</td>
</tr>
</tbody>
</table>

Figure 47 Example of enriching a dataset with temperature data

Challenges for data enrichment include: Finding, understanding, accessing useful third-party data to use for enrichment; Often join cannot rely on shared identifiers between the left and the right table; Values on the left and in the right table need to be reconciled to evaluate the join condition; The user may be expert in his domain and familiar with the left table, but little familiar with the data in the right table; The user may need to enrich a very large volume of data (e.g. ~ 500M rows); Competency gap: data scientists ≠ data engineers

A.1. Data Enrichment Evaluation Set Up

The goal of the evaluation task is to achieve the same output of the EW-Shopp tutorial given at the ESWC conference in 2019 (done with DataGraft) through different software. For the EW-Shopp tutorial the pipeline represented in Figure 48 was used to enrich the initial dataset with weather data through DataGraft. We can generalize this pipeline in Figure 48 with the following activities:

1. Data transformation (format data)
2. Recognize the city in GeoNames (reconciliation, instance-based matching, merge dataset)
3. Extension with ADM1, ADM2, ADM3 (Administrative Divisions level 1/2/3 - i.e., region, province, commune)
4. Data Transformation (filter, group by)
5. Enrich data with ECWMF data
6. Download CSV

The software used for the analysis are the following: Trifacta, Knime, Talend and the goal of our evaluation is to compare different software on the defined pipeline in order to evaluate them and understand strong and weak points of the tools.
A.2. Comparison to Trifacta

Trifacta is a web application for data transformation. It doesn’t require download and it has three different versions: Free, Pro (419$/month), Enterprise. For our evaluation we used a trial version which had many features of the Enterprise version. Trifacta has a partnership with Google and in 2019 they launched Cloud DataPrep, using Trifacta web app as basis.

A.2.1. Realization of the Pipeline Steps

It is possible to use external APIs in the Enterprise version with User Defined Function (UDF), but it must be implemented in an IDE. Unfortunately, is not possible to use an IDE in the trial version. So, for our case it was not possible to connect Trifacta to GeoNames since it was not possible to use the IDE (e.g., through an UDF it would be possible for example connecting Trifacta to GeoNames through a reconciler).

We proceeded downloading the dump of Germany from GeoNames, this was possible only because we already knew that there were only German cities in the initial dataset, we filtered cities to the category ADM4 (administrative division 4), also this step was possible only because we already knew from DataGraft experience that the cities in our dataset fit well with the ADM4 category.
After these pre-processing on the GeoNames dump we proceeded with a left join between the 2 files, in this way we were able to enrich the initial dataset with the information from the GeoNames dump.

### A.2.2. Joining Datasets

Instance-based matching feature doesn’t exist in Trifacta, but there are different properties used to improve a simple join Figure 49.

We tried to use the Fuzzy Match with a left join, but from input 682 rows the result consisted of 1194 rows! Some cities of the initial dataset were matched with multiple cities through a metaphone function (e.g. Aalen and Wollin were evaluated similarly)

So, we proceeded without the Fuzzy match option since it was a binary option with no customizable settings.

![Figure 49 join feature in Trifacta](image)

The results of the enrichment will be described in the paragraph A.5.

### A.2.3. Discussions and Conclusion

Trifacta has a simple and intuitive UI, the dataset showed to the user changes according to the steps done in the pipeline, in this way the user can keep track of the changes and work directly on the tabular form Figure 50, moreover Trifacta tabular functions seem intuitive and require only basic coding abilities.
Trifacta samples files bigger than a certain dimension (information on threshold is not public but seems to be around 1MB) in order to process them faster, the user chooses the sampling methodology he/she prefers: Figure 51.

Trifacta is an effective tool for tabular transformation, in fact we were able to bypass some of its limitations and carry out all the tabular transformation easily but only a part of the reconciliation & extension activities (e.g. join cities). It will be interesting to replicate the pipeline using User Defined Function (UDF) by upgrading the Trifacta version with the Google CloudPrep trial version.
One solution that could be implemented by a user (through the User Define Function) is to enhance Asia with a java function, in this way it would be possible to reconcile data, connect to web services and overcome many of the problems found in this experiment.

Trifacta struggled with tasks that required a connection to a webservice: GeoNames and Weather extensions, the join was not accurate enough to obtain the same level of the reconciliation as with DataGraft.

With a greedy iteration it was possible to extend information from ADM4 to ADM2 and ADM1 thanks to GeoNames dump documentation, but It may not be possible to do the same with other webservices different than GeoNames. In conclusion the Pipeline required for the Dataset “EW-Shopp tutorial” was carried out only partially by Trifacta since it was not able to extend on ADM3 but mostly because it was not possible to enrich data with weather information.

Moreover, since it was carried on with a “greedy” methodology there is no evidence that this method could be suitable for similar task with different pipelines or dataset.

### A.3. Comparison to Knime

KNIME stands for Konstanz Information Miner, is a free desktop software for analytics with a modular platform for building and executing workflows using predefined components called nodes. There is a premium version of KNIME: KNIME Server is a service that deploys the software to a webservice and is available through an annual subscription.

Knime core functionalities are used for tasks such as standard data mining, analysis and manipulation and these features and functionalities can be extended through extensions from various groups and vendors. Moreover, KNIME has a vast community and content documentation such as videos, forum, tutorials which helps the user to better learn the tool.

#### A.3.1. Realization of the Pipeline Steps

In KNIME we used different SPARQL queries to download ADM1, ADM2, ADM3, ADM4 data from GeoNames, we then merged these results in order to enrich ADM4 with higher geospatial level (ADM4->ADM3->ADM2->ADM1)

We then applied an Instance-based match (more information on the join in next paragraph) between our initial dataset and the GeoNames data. This way we were able to reconcile the city of our initial data with ADM4 and enrich the initial dataset with more geospatial information such as ADM3, ADM2, ADM1.

After some tasks of data transformation such as filter, group by, format correction, we enriched the dataset with meteorological data through ECWMF with a POST request having the date and the city of interest of each row as input.

We then transformed the json data in CSV format and downloaded the result.
A.3.2. Joining Datasets

The instance-based join was performed through a string matcher node (see Figure 53).

Figure 52 Screenshot of the KNIME pipeline

Figure 53. Dialog - 2/18: String Matcher
A.3.3. Discussions and Conclusion

Through empirical tests we decided to keep as threshold distance $= 2$ since from this point the join started to be less reliable and the rate of wrong matches was increasing.

In KNIME the pipeline is reusable for different files but the KNIME free version is not scalable, so it can be difficult to replicate the results for larger dataset. KNIME offers a great variety of tools (Machine learning, SPARQL, statistics) and if needed it is possible to add scripts in different coding languages such as Python, R, Java, JavaScript or MATLAB. KNIME’s flexibility was the key element to solve this pipeline, the same pipeline that Trifacta partly failed. As for the negative side, the number of nodes (steps) used to obtain this flexibility was much higher than in DataGraft.
In KNIME The SPARQL query was possible only thanks to an endpoint made available by UNIMIB, GeoNames doesn’t have its own SPARQL endpoint. KNIME is the only tool with a SPARQL function in its functionalities. The weather extension through the POST request to ECWMF was possible only through a service offered by UNIMIB. Through KNIME Server an organization could perform data mining and data enrichment with a unique tool with no problem of scalability, KNIME was also able to deliver a comparison metric for the string distance in city reconciliation which discovered the presence of homonyms in GeoNames ADM4. In conclusion, KNIME has shown a great flexibility reaching the goal of the pipeline and successfully enriching the cities with weather data.

### A.4. Comparison to Talend

Talend is an Eclipse-based visual programming editor. Similar to KNIME, Talend uses predefined components called nodes in order to create a pipeline, this pipeline generates an executable Java code.

Talend is an open source data integration platform, there are different version available, two of the most popular are the free of charge desktop version (Open source) and the Talend Cloud data integration tool (monthly price= 1.170$/usd)

### A.4.1. Realization of the Pipeline Steps

Repeating the same process of Trifacta and KNIME would have not helped us to gain more knowledge of the process since the steps required would have been the same.

With Talend we tried a different path to retrieve weather information, not through GeoNames but enriching directly through the attribute “city name” with OpenWeather through a POST request.

It is possible to reconcile with OpenWeather through different attributes:

- city name
- city ID
- geographic coordinates
- ZIP code

Initially it didn’t seem possible to reconcile with OpenWeather through ADM1, as requested by the tutorial pipeline, but with further analysis we discovered (it is not documented in any OpenWeather documentation) that “city ID” is the GeoNames ID. So, it would have been possible to enrich our dataset with ADM1 weather information.

However, we proceeded to enrich our dataset with weather information through the city attribute, we used a GET request to OpenWeather from the initial dataset, in this way we obtained a JSON file with the weather information for each row, we then transformed the JSON file in a CSV format and we joined it to our initial dataset.
A.4.2. **OpenWeather**

OpenWeather is a service similar to ECWMF that offers weather data. It has different pricing and for our analysis we used the free version, but in this way it was not possible to retrieve historical data.

A.4.3. **Discussions and Conclusion**

The UI structure is similar to KNIME but with different functionalities, Talend is more focused on the data integration. The element that distinguishes this test from the previous ones is the use of OpenWeather. OpenWeather can be a competitor of ASIA functionality on the matter of Weather enrichment, since ECWMF and OpenWeather retrieve similar data, in this case a final user might prefer a tool depending on the price of the information.

In our pipeline, OpenWeather was able to enrich the dataset with weather information using the city name without an intermediate connection (e.g. GeoNames). Although OpenWeather didn’t take a date as input (premium service required) or an ADM1 it proved that it can be used to enrich geospatial information with weather data and it doesn’t seem to have the problem of homonyms.

A.5. **Conclusion**

Starting from 682 rows, the results with:

- DataGraft are: 675 matched joins, 7 missing joins
- Trifacta are: 591 matched joins, 119 missing joins, 28 homonym joins
- KNIME are: 656 matched joins, 26 error joins*, 30 homonym joins
- Talend/OpenWeather are: 660 matched joins, 22 missing joins.
error joins exist only in KNIME because it allows the user to analyse the result with a metrics of confidence

Figure 56 is a first representation of city reconciliation results with different tools, the “Matched” value contains what the tool returned to the user (contains both correct and incorrect matches), the “Missed” value contains the number of city that the tool was not able to reconcile, the “Error” value contains cities that for sure we knew that were incorrect matches (only in KNIME thanks to the string matcher node), the “Homonyms” value contains the number of cities that were reconciled with two different cities with the same name.

After this first analysis we performed a deeper one, in which we manually, one by one reconciled the cities in our dataset to GeoNames. From 682 rows of our initial dataset there were 192 unique cities.

Since Talend+OpenWeather reconciled the cities with different entities (ADM3, PPL, ADM2 etc.) we were not able to verify its correctness, so we performed a city enrichment only for Talend following the steps of the Trifacta pipeline; the results of this analysis are showed in Figure 57.

(here [https://public.tableau.com/profile/shady7269#!/vizhome/ReconciliationEvaluation/Dashboard1](https://public.tableau.com/profile/shady7269#!/vizhome/ReconciliationEvaluation/Dashboard1) can be found the higher resolution version of Figure 57).
Correct reconciliation: cities that were correctly reconciled to GeoNames
City ambiguity not recognized: cities in the dataset that should have reconciled to two or more cities in GeoNames (e.g., homonyms) but are reconciled to only one city
City ambiguity recognized: cities in the dataset that are correctly reconciled to two or more cities
Incorrect reconciliation: cities that are reconciled to a wrong city
Missing reconciliation: cities that have not been reconciled to a city

This analysis shows that DataGraft has the highest value of correctly reconciled cities, but it lacks in the recognition of the ambiguity (homonyms). Trifacta has the highest number of cities correctly recognised and the lowest of incorrectly reconciled but has also the highest number of missing reconciliations. KNIME and Talend both have a high number of incorrect reconciliations.

### A.5.1. Comparison

To compare data enrichment tools for the EW-Shopp project, we looked at tools from 4 different points of view:

1. Main characteristics: attributes that help to describe the main activities of a software in the field of data enrichment (e.g. the possible transformation of the data set)
2. Other characteristics, attributes not mandatory for a data enrichment tool but that can help with the scope (e.g. sampling, RDF mapping)
3. Structure, attributes that describes the structure of the software (e.g. requires installation)
4. Other Information, important but nonrelevant for practical activities (e.g. price, documentation)
### Main activities for Data Enrichment

<table>
<thead>
<tr>
<th>Tool</th>
<th>Tool purpose</th>
<th>Join Table</th>
<th>Instance based matching</th>
<th>User-free Reconciliation</th>
<th>Data extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagraft</td>
<td>Data Enrichment</td>
<td>Only through RDF mapping</td>
<td>Yes, in Asia*</td>
<td>Yes, for a set of services</td>
<td>Yes</td>
</tr>
<tr>
<td>Trifacta</td>
<td>Data preparation</td>
<td>Yes</td>
<td>Fuzzy match (metaphone function)</td>
<td>No</td>
<td>Not by default</td>
</tr>
<tr>
<td>Knime</td>
<td>Data mining</td>
<td>Yes</td>
<td>String distance comparison node</td>
<td>No</td>
<td>Yes, with an GET request</td>
</tr>
<tr>
<td>Talend</td>
<td>Data Integration</td>
<td>Yes</td>
<td>Fuzzy Match (metaphone and levenshtein function)</td>
<td>No</td>
<td>Yes, with a GET request</td>
</tr>
</tbody>
</table>

**Table 12: main activities for data enrichment**

Here is a description for the metrics considered for the main activities:

**Tool purpose:** Which is the main purpose of the tool? (Machine learning, tabular transformation, RDF mapping)

**Join table:** The tool supports (left, right, outer inner) join between two tables?

**Instance based matching:** Does the tool support the join between similar values user-free?

**Reconciliation:** The tool can link values that occur in the table to identifiers in external knowledge basis with no knowledge of the external knowledge basis by the user?

**Data extension:** The tool allows the user to extend the initial dataset from the attributes of the reconciled one?

### Other characteristics

<table>
<thead>
<tr>
<th>Tool</th>
<th>Custom tabular transformation</th>
<th>RDF mapping</th>
<th>SPARQL query</th>
<th>UI results</th>
<th>Use of Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagraft</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Trifacta</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Knime</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Talend</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Table 13: Other characteristics**

Here is a description for the metrics considered for the other characteristic:

**Custom tabular transformation:** It is possible to create ad hoc tabular transformation?

**RDF mapping:** It is possible to perform an RDF mapping through the tool?

**Sparql query:** It is possible to perform a SPARQL query through the tool?

**UI results:** The results are displayed in real time in an interface table during the activities?

**Use of Pipeline:** The tool uses pipelines?
Table 14: Structure

Here is a description for the metrics considered for the structure:

**Application type:** Is the tool a web application? Does it require a download?
**Input:** Which files can be used as input?
**Output:** Which files can be downloaded as output?
**Scalability:** The tool can perform the activity with a dataset of different dimension?

<table>
<thead>
<tr>
<th>Tool</th>
<th>Application type</th>
<th>File input</th>
<th>Output</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagraft</td>
<td>Webservice, no download</td>
<td>CSV</td>
<td>RDF, CSV, JSON, JAR file</td>
<td>Yes</td>
</tr>
<tr>
<td>Trifacta</td>
<td>Webservice, no download</td>
<td>CSV, JSON, TSV, Avro, LOG, Parquet</td>
<td>CSV, Trifacta (own format), Log</td>
<td>Yes</td>
</tr>
<tr>
<td>KNIME</td>
<td>Free desktop software, or Premium webservice</td>
<td>CSV, JSON, XML, LOG, TSV and other not specified</td>
<td>CSV, JSON, XML, LOG, TSV and other not specified</td>
<td>Yes through Premium webservice</td>
</tr>
<tr>
<td>Talend</td>
<td>Free desktop software, or Premium webservice</td>
<td>CSV, JSON, XML, LOG, TSV and other not specified</td>
<td>CSV, JSON, XML, LOG, TSV and other not specified</td>
<td>Yes through Premium webservice</td>
</tr>
</tbody>
</table>

Table 15: Other information

Here is a description for the metrics considered for other information:

**Online Documentation:** How can the users learn or ask questions about the tool?
**Custom code:** Is it possible to add custom script in the tool?
**Price:** What is the price of the tool?
**API:** Is the tool reachable through an API?

<table>
<thead>
<tr>
<th>Tool</th>
<th>Online Presence</th>
<th>Customization</th>
<th>Price</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datagraft</td>
<td>Documentation</td>
<td>IDE presence (Java), flexible customization</td>
<td>400€/month per user (The free version is very limited)</td>
<td>Yes</td>
</tr>
<tr>
<td>Trifacta</td>
<td>Documentation, Forum, Youtube tutorials</td>
<td>Yes (Python, R, Java, Javascript, Matlab)</td>
<td>Free or premium version with different prices</td>
<td>Not in the free version</td>
</tr>
<tr>
<td>KNIME</td>
<td>Documentation, Forum, Youtube tutorials</td>
<td>Yes (Java)</td>
<td>Free or premium version 1170$/month per user</td>
<td>Not in the free version</td>
</tr>
<tr>
<td>Talend</td>
<td>Documentation, Forum</td>
<td>Yes (Java)</td>
<td>Free or premium version 1170$/month per user</td>
<td>Not in the free version</td>
</tr>
</tbody>
</table>

**A.5.2. Discussion on the Pipeline**

The pipeline performed has two macro activities:
1. Tabular transformation (e.g. group by, filter, order, string editing)
2. Data enrichment (e.g. enrichment with weather data)

3. The tabular transformation part (1) is already developed and structured by different tools and it doesn’t seem to be possible to overcome those tools' performance (e.g. Trifacta is a whole organization focused on data preparation).

4. (The RDF mapping is not supported by all the tools except DataGraft, this can give an advantage to DataGraft methodology in terms of mapping since it considers the semantic relations, although the RDF mapping was not strictly requested by this task since input/output are CSV files but it could have improved the output accuracy.)

5. On the contrary, the enrichment part (2) is usually delegated to the user and he/she must know the structure of the external knowledge graph

6. As shown in the OpenWeather example, taken individually an ASIA enrichment task can be replicated (or overcame) by a single service with a certain level of difficulty. However, this level of difficulty increases with the number of external services desired. Moreover, the price and the subscription to services could easily lead to a great cost since every service has its own fee (e.g. OpenWeather).