TERRITORI APERTI

FONDO TERRITORI LAVORO E CONOSCENZA CGIL, CISL, UIL

Deliverable D7.1

Territori Aperti: Tecniche automatiche di raccolta e preparazione dei dati

http://territoriaperti.univaq.it

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Project Title	:	Territori Aperti

Deliverable Number	:	D7.1
Title of Deliverable	:	Territori Aperti: Tecniche automatiche di raccolta e preparazione dei dati
Nature of Deliverable	:	Report, Other
Dissemination level	:	Public
Licence	:	–
Version	:	0.1
Contractual Delivery Date	:	
Actual Delivery Date	:	
Contributing WP	:	WP1 WP3
Editor(s)	:	Antinisca Di Marco (UNIVAQ), Amleto Di Salle (UNIVAQ), Giordano d'Aloisio (UNIVAQ)
Author(s)	:	Antinisca Di Marco (UNIVAQ), Amleto Di Salle (UNIVAQ), Giordano d'Aloisio(UNIVAQ)
Reviewer(s)	:	Antinisca Di Marco (UNIVAQ), Amleto Di Salle (UNIVAQ), Giordano d'Aloisio (UNIVAQ), Giovanni Stilo (UNIVAQ)

Abstract

This deliverable describes the goal, the architecture and the V1 implementation of TA-Analytics, a system for collecting and analyze Open Data.

Keyword List

Open Data, Data Analytics, Data Collection, Data Wrangling, Business Intelligence, Open Repository

Glossary, acronyms & abbreviations

Item	Description
D.I.	Data Importer
D.A.V.	Data Analytics and Visualization
B.I.	Business Intelligence
T.A.	Territori Aperti

Table Of Contents

List Of Tables

List Of Figures

1 Introduction

One of the key goals of the Territori Aperti (TA) project is to manage and extract knowledge from several sources of information. Thus, being able to collect, process and analyse different types of data is a grounded aspect of it.

From an abstract point of view, to achieve the aforementioned result, a dataset must be processed and properly presented to the end-user.

Processing a dataset implies manipulation of it by applying various techniques as - but not limited to - data cleaning, data wrangling and sampling. Furthermore, many analysis techniques - as machine learning or descriptive analysis - can be applied at any stage of the processing pipeline. The results of the processing and analysis phases can be presented to the user using a proper succinct graphical representation (plots, charts and diagrams).

Data are heterogeneous in their nature, in terms of content and of sources, making it difficult to apply the process, described above, in a standardised way. Moreover, to provide useful information to their users, the content provider (mainly public organisations) are continuously updating the data making the depicted scenario much more difficult.

To allow TA project users to access the available data, analyze and visualize them, we present the TA-Analytics platform, a system for collecting and analyzing Open Data coming from different open repositories. We like to remark that TA-Analytics follows the principles of Open Science at the basis of Territori Aperti. Moreover, the TA-Analytics allow the user to visualise the gathered data using different types of charts and finally share their analysis with other users (as citizens or public administrations which are not directly involved in the TA project).

TA-Analytics also have the capability to interact, on a periodic base, with several open repositories, which exposes their data by web services. Lastly, users are able, accordingly with the Open Science principles, to upload their own open datasets to share them with the other platform users.

This report is organized as follows:

- **Chapter 2** describes the overall architecture, his components and use cases and the functional and non-functional requirements considered for building the system.
- **Chapter 3** describes the first implementation of the system. Each section is dedicated to a particular component and describes the technologies and the design decision taken.
- **Chapter 4** concludes the document describing the future steps and improvements for the system.

2 System Architecture

TA-Analytics in its essence can be seen as a system made by two macro components that performs respectively two macro tasks:

1) Download data from external open source repositories and make them available to the users.
2) Allow also non-expert users to manipulate the data and extract information from them in an easy way.

This tasks contain several critical aspects that have to be considered for the application to really be useful and satisfy the users needing. So first of all we have made an analysis of the requirements in order to better organize the development of the system.

In the next sessions we will describe more in detail this process starting from the Functional and Non-Functional Requirements (Section: 2.1) and the relative Use Cases (Section: 2.2) of the system to then describe his architecture and his macro components (Section: 2.3).

2.1. Functional and Non-Functional Requirements

Table 2.1 describes the Functional Requirements of the system. Requirements from FR1 to FR3 concern the process of downloading Open Data from different web repositories and make them always updated and available to the users. Requirements from FR4 to FR6 instead concern the other macro task of TA-Analytics that is the analysis and visualization of data by the users. Users must be able to use the data imported by system and must be able to share their analysis to other users and in other platforms. They must also be able to upload their own Open Data and share them with other users thus reflecting the principle of sharing knowledge that is at the basis of Territori Aperti.

Table 2.2 describes instead the Non-Functional Requirements of the system. NFR1 exposes a very critical requirement since all the users must be able to do analysis and manipulate the data without knowing specific languages for data manipulation or knowing at most only the basis. NFR2 and NFR3 concern instead the data importing process. Since this process is done automatically, the system must be able to handle any kind of error that could happen in a proper way and must be able to manage any kind of dataset of any format and easily integrate new data sources.

FR1	The system must be able to interact with different web services related to Open Data repositories for downloading the data.
FR2	The system must be able to update periodically the data he gets from open repositories in order to make them always updated.
FR3	The system must store the downloaded data in order to make them available to the users.
FR4	The system must allow the users to make analysis and charts with the data at its disposal.
FR5	The system must allow the users to create interactive dashboards with charts and analysis and to share them with other users.
FR6	The system must allow the users to upload their own datasets in the form of CSV or XLS files and share them with the other users.

Table 2.1: Functional Requirements

NFR1	The system must be easy to use also for non-technical users.
NFR2	The system must be able to automatically recover in case of an error during the automatic download of data.
NFR3	The system must be able to download and manage datasets of different formats and easily integrate new data sources.

Table 2.2: Non-Functional Requirements

2.2. Use Cases

Figure 2.1 describes the possible use cases of the system. We have three kinds of actors: two human and one non-human. Talking about human actors we distinguish between Registered User and Non-Registered User: for Registered User we intend any user that is actually registered (via a proper authentication system) in the application and can access his functionalities, instead with Non-Registered User we identify any other kind of user that is not registered in the system. Only Registered Users can create dashboard and analysis: creating a dashboard includes the process of creating any kind of data visualization (for example graphs or tables) that will then be included inside the dashboard; all the data visualization models, for a specific dataset, has to be included inside a dashboard in order to be shared with other users. For this reason the Create Dashboard use case includes the Create Data Visualization use case and a Registered User must always pass through the Create Dashboard use case in order to create a data visualization. Instead a Registered User does not always have to a create a dashboard in order to create an analysis, in fact a user can create an analysis and put the results inside another dataset that will then be shared to other users without inserting it inside a dashboard. For this reason the Create Analysis use case is not included in the Create Dashboard use case but is only extended by it indicating that not always an analysis is required to create a dashboard. A Registered User can also upload his own datasets and share them with other Registered Users and can download datasets available to him from the system.

A Non-Registered User instead can only use public dashboards created by Registered Users: a public dashboard is a specific kind of dashboard that can be accessed also outside the system embedding it for example in other applications or web sites. For *using a dashboard* we mean to watch the data representation in it and eventually filter the data if some kind of filtering is present in the dashboard. A Non-Registered User can not change the content of a dashboard in terms of charts and datasets inside it. From a public dashboard is also possible (if allowed) to download the datasets used in it, so the Use Public Dashboard use case extends the Download Data use case. Obviously a public dashboard can also be used by a Registered User.

Finally there is an actor representing the Data Importer (D.I.) application which periodically downloads data from different open repositories and stores them inside the system. We decided to treat the D.I. app as an actor to highlight the process of automatic download and storage of Open Data.

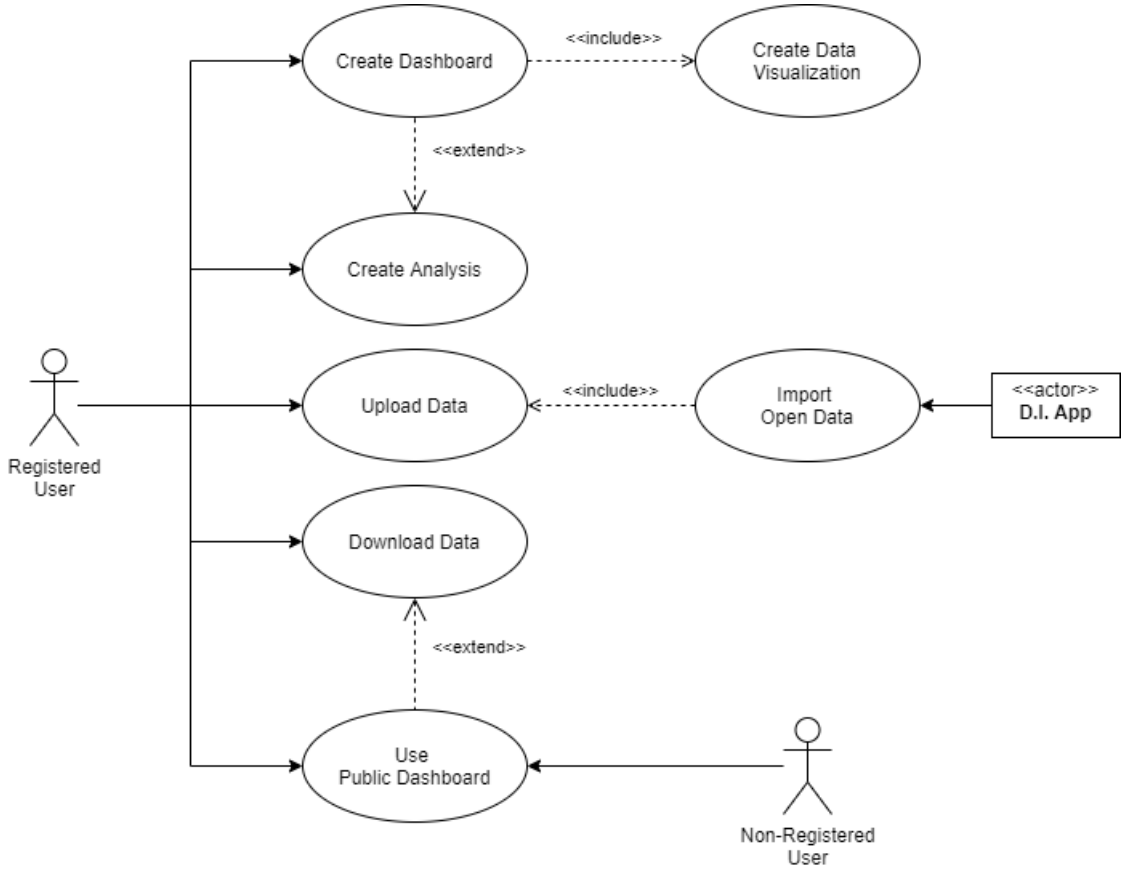

Figure 2.1: TA-Analytics Use Case Diagram

2.3. System Components and Interactions

The main components of the system are shown in figure 2.2. As can be seen, the system is made by two principal components which interacts with the interfaces exposed by the others.

The first is the Data Importer (D.I.) component which is the one responsible of downloading periodically datasets from different Open Data repositories interacting with the services exposed by them. The component has to be able to interact with different web services using different protocols if necessary. The D.I. application interacts also with a database for storing the imported data. Figure 2.3 describes more in detail the process of importing open data performed by the component: first the D.I. makes a request to all the data repositories requesting their datasets, then, after the data repositories have sent their data, it performs some data processing if needed in order to finally store the data in a database. This process is repeated periodically in order to have always the datasets updated.

The second main component is the Data Analytics and Visualization (D.A.V.) application which also interacts with the database to retrieve the downloaded datasets. The D.A.V. component is the entry point for the end user to all the datasets and services offered by the application, allowing him to build analysis, charts and dashboards, to upload his own datasets and to download the datasets available to him. This component will offer to the users a graphical interface to interact with the data, allowing them to make simple analysis abstracting from the data manipulation languages used by the application. Figure 2.4 describes the process of creating a dashboard by a Registered User: first the user makes a request to the system for creating a dashboard, then the application asks to user to select a specific dataset to be included inside the dashboard and, after querying it from the database, asks to the user which kind of visualization he wants to include inside the dashboard for this specific dataset. These two processes (selecting a dataset and selecting a visualization) can be done several times, meaning that a dashboard can include different datasets and for each of them different kinds of visualizations. Finally,

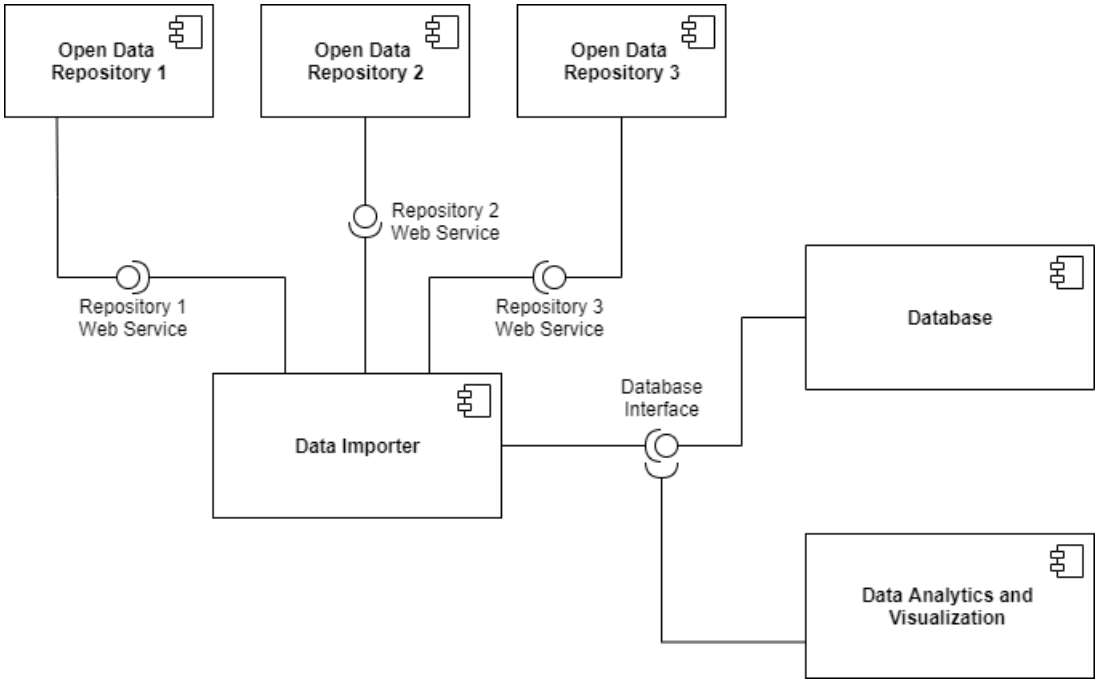

Figure 2.2: TA-Analytics Component Diagram

after the user has populated the dashboard with datasets and visualizations, he publishes it making it available to the other Registered Users or, if the dashboard is public, also to Non-Registered Users.

This separation of concerns between the D.I. and D.A.V. components allows us to modify or completely change if necessary one of these components without affecting the behavior of the other ensuring modularity and scalability to the system.

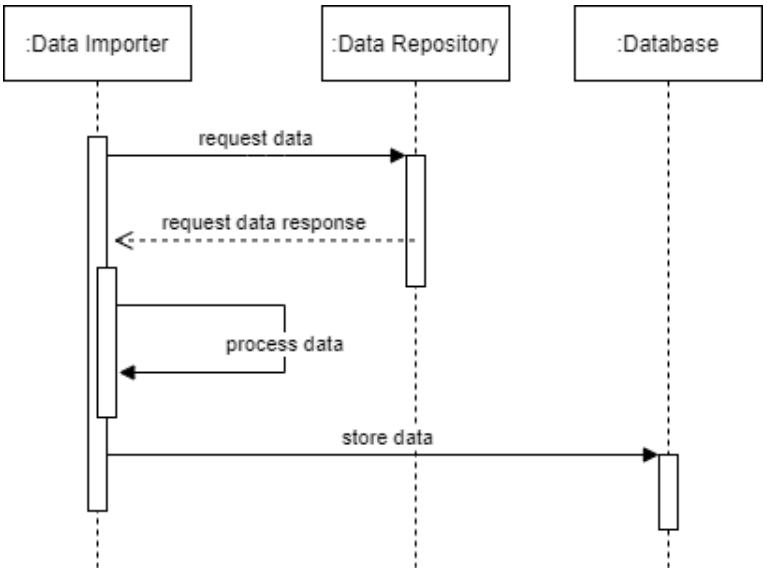

Figure 2.3: Import Open Data Sequence Diagram

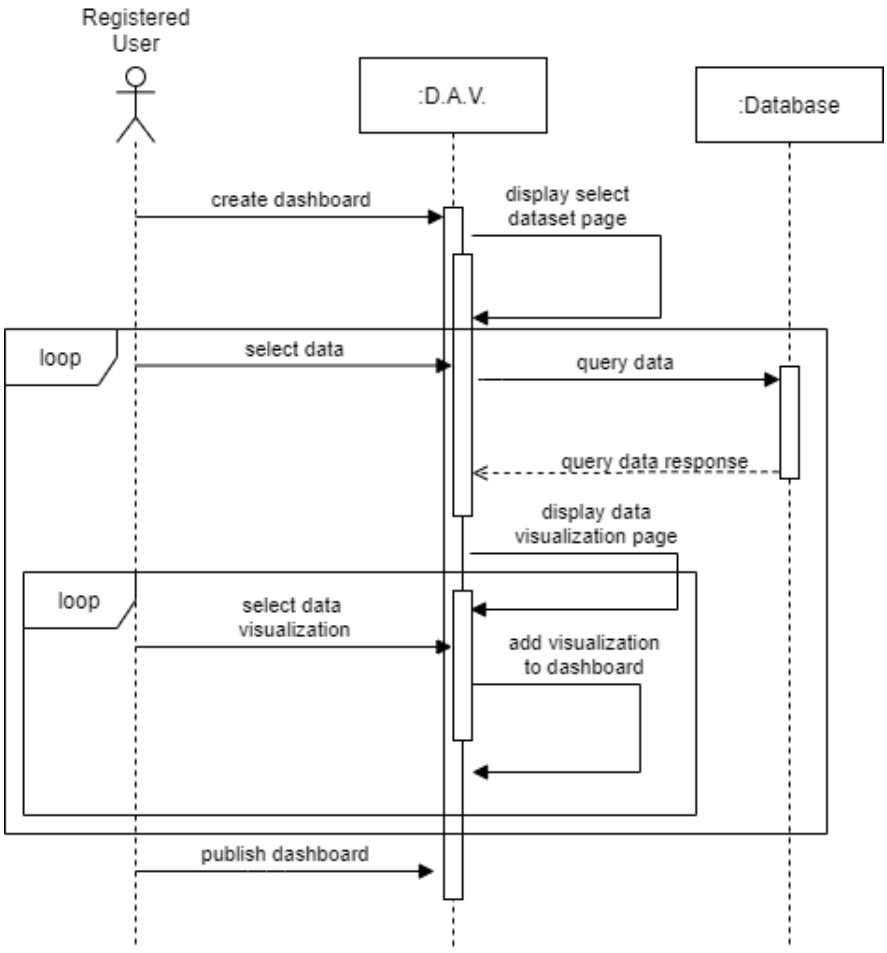

Figure 2.4: Create Dashboard Sequence Diagram

3 TA-Analytics implementation version 1.0

This chapter describes the first implementation of TA-Analytics from a more technical point of view. An informal overview is shown in figure 3.1. In this first release the system only retrieves data from the I.Stat[1] repository using the SDMX protocol (section 3.1.1) to interact with his web services. After it, the Data Importer (D.I.) application stores the datasets into a SQL database. Since all the data managed by the application are structured data, we decided in this first release to use a relational database instead of a NoSQL one. Finally, there is the Data Analytics and Visualization (D.A.V.) app which is the entry point for the end user to all the datasets stored by the application. With this component the user can analyze the available data and build interactive dashboards which can then be exported and embedded in other web applications. In the next sections we will describe more in detail the implementation of these components and the design decisions made while developing them.

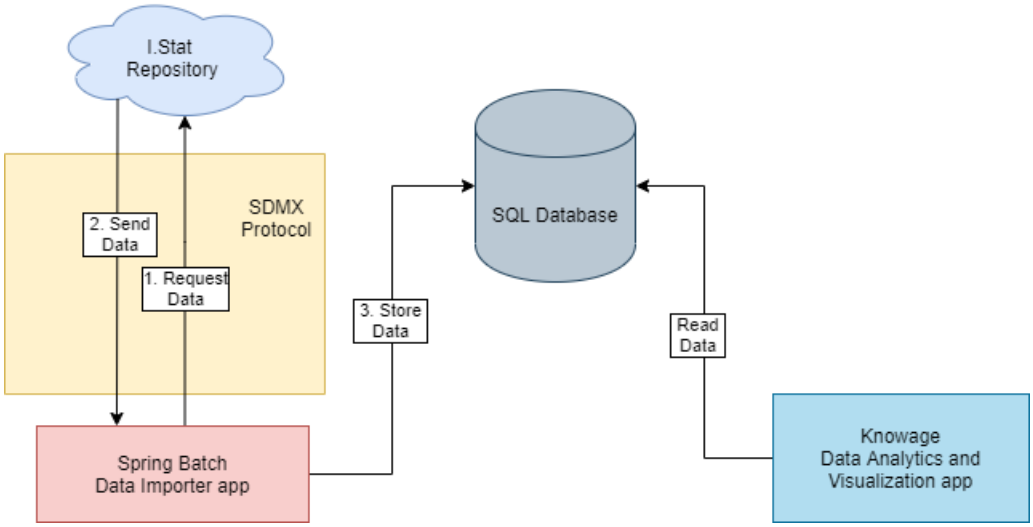

Figure 3.1: TA-Analytics V1 Components Overview

3.1. Data Importer

The Data Importer is the component responsible of retrieving periodically data from different open repositories interacting with their web services. The web services with which the system can interact can be multiple and heterogeneous in terms of implementations and protocols used so a lot of care has been given to the development of an application able to interact easily with different external services using different protocols and messages. The application has been built in Java using the Spring Boot framework, which eases a component based developing approach and offers an easy integration with external libraries and services. A more detailed overview of the internal components of the D.I. application is shown in figure 3.2 while the entire batch process of downloading and store the datasets is shown in figure 3.3. This process can be described with the following steps:

[1] http://dati.istat.it

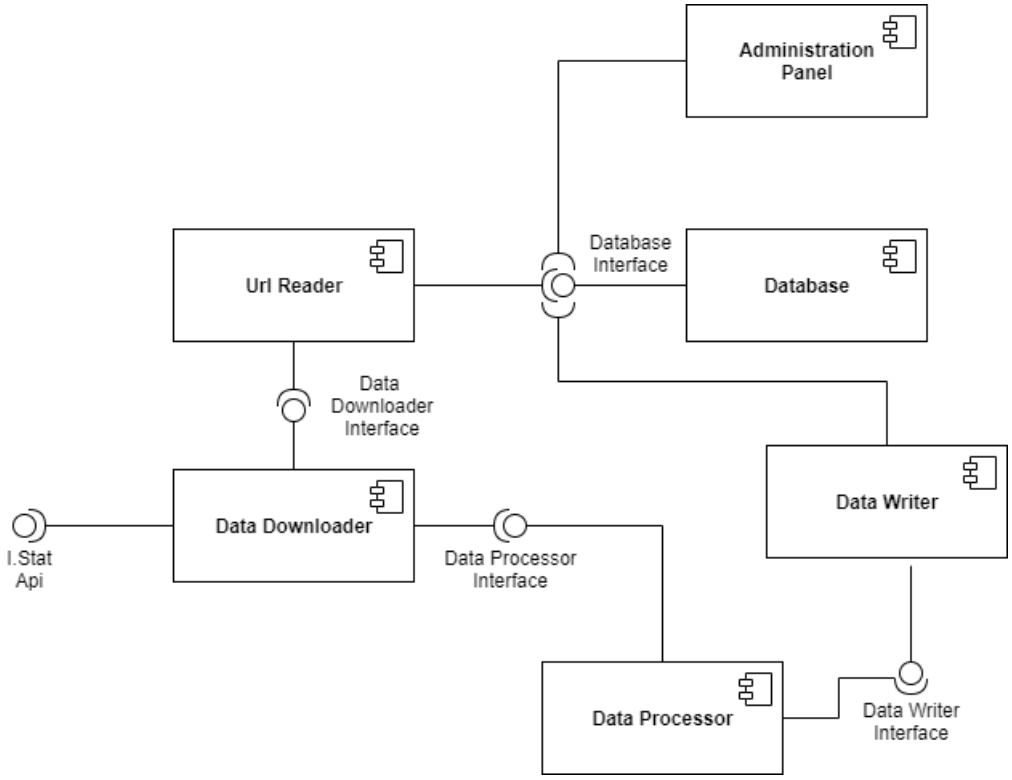

Figure 3.2: Data Importer Components

1) The Url Reader component reads from the database the urls corresponding to the API calls that has to be made to the I.Stat service in order to have the datasets.
2) The Data Downloader component uses these urls to make a remote call to the I.Stat service to retrieve the datasets needed.
3) Since the datasets returned by the I.Stat repository are represented using the SDMX format, the Data Processor component has the task of converting these data in a format that can be stored inside a SQL database.
4) Finally the Data Writer component stores the processed data in the database.

The entire batch process has been implemented using Spring Batch which is a Spring sub-framework designed to enable easily the development of robust batch applications, offering features like error management, job restarting, job skipping, logging and so on [1].

The system has also a user interface from where an administrator can insert new urls corresponding to new datasets or manually start the above process if needed.

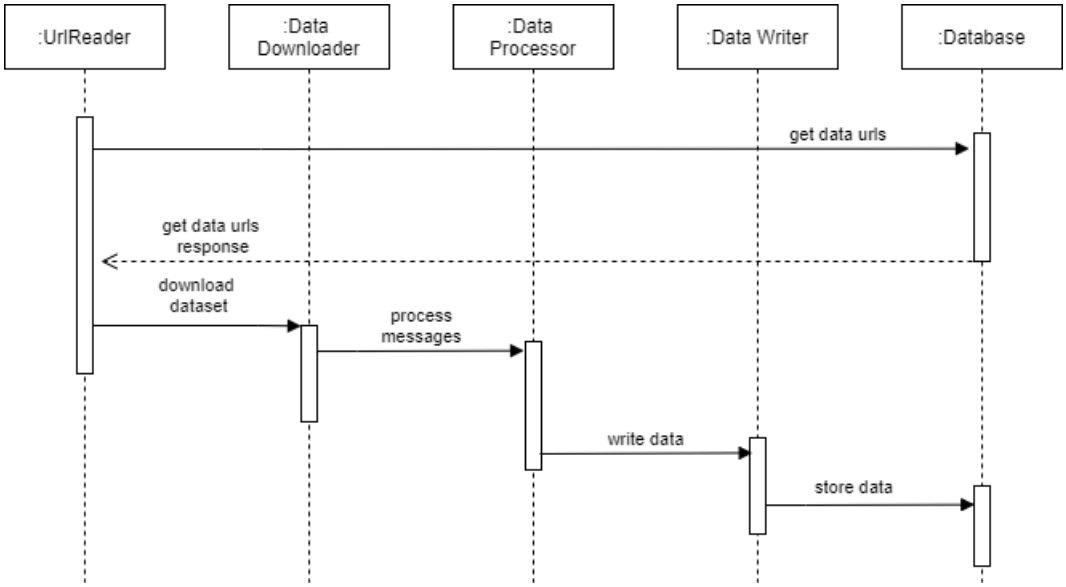

Figure 3.3: Data Importer Sequence Diagram

3.1.1. SDMX Protocol

As mentioned before, for this first release the application interacts with the I.Stat repository, that is one of the biggest open data repositories in Italy. I.Stat exposes a REST API for interacting with external applications and data are sent using the SDMX protocol, which is a standard protocol for sending statistical data.

Figure 3.4 shows the SDMX Metamodel. As can be seen from the picture, this model provides a lot of components for modeling statistical datasets: the two most important are the Dataflow and the Data Structure Definition (DSD) components which represent respectively the data and the structure of a dataset. The Dataflows are grouped in different Categories and each Dataflow refers to a single DSD. So, a Dataflow can have only one schema associated and must respect the constraint imposed by that structure, instead a Data Structure Definition can be associated to more dataflows (for example corresponding to different observations during the time). The DSD is made by different Codes and Concepts which are grouped inside Code Lists and Concept Schemas. The Concepts are the attributes of the dataset, while the Codes are a list of possible values that a Concept can have (e.g. the attribute SEX can be a Concept of a DSD while Male and Female are the Codes related to that Concept). Finally, there is the Provision Agreement, that is the contract made by the Data Providers (the organizations that provide the datasets) to their clients regarding the dataflows they can provide. Each Dataflow is identified by an ID and a version unique for his Data Provider.

Figure 3.5 describes more in detail the representation of information inside a SDMX dataset. A dataset must reference to a DSD either directly or via a Dataflow or Provision Agreement. Logically a dataset comprises Series and the Series comprises Observations. Observations are associated to a Time Period for time series or to any other Dimension (e.g. geography) for non-time series. Attributes can relate to the dataset, the Series, an Observation or to a Group. A Group is a partial series key i.e. the value of one or more Dimensions comprising a sub set of the full key. Depending on the physical format of the datasets the associations shown by the green lines can be either containers (e.g. Dataset contains Series) or references (Series referenced the Dataset).

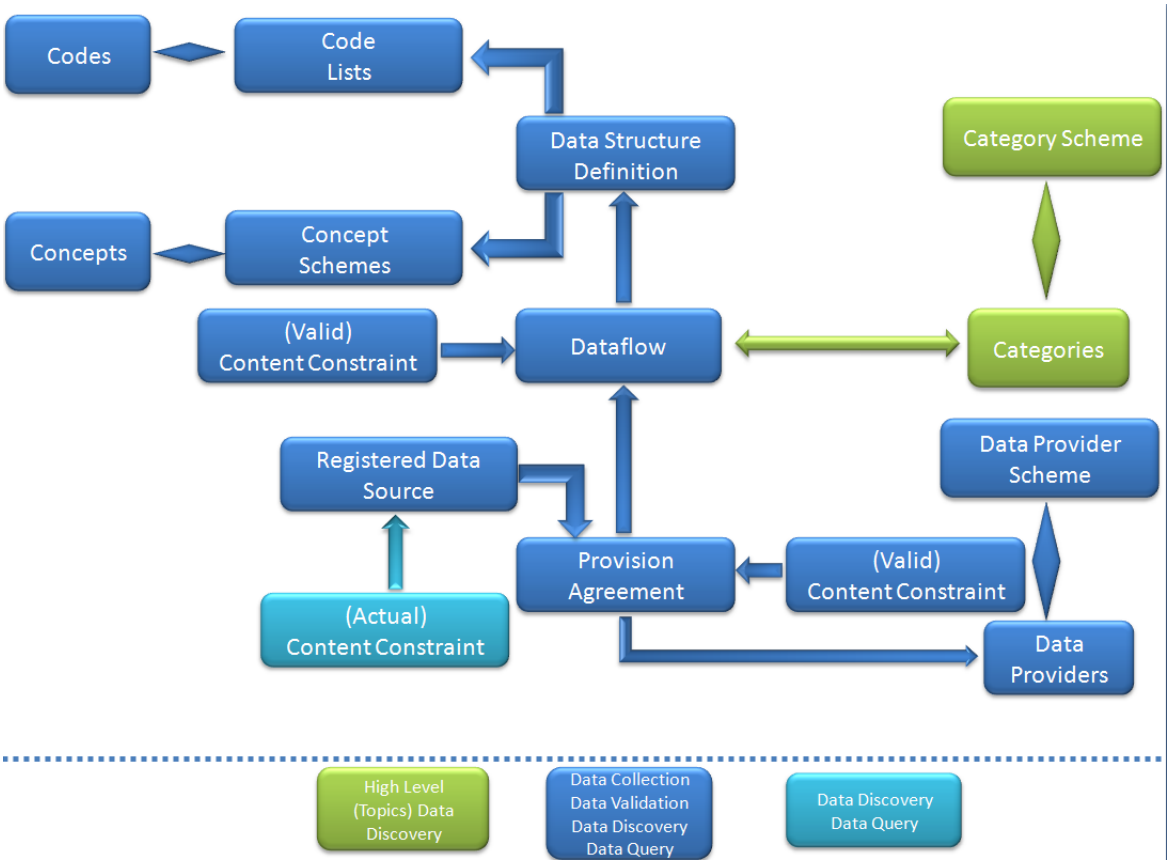

Figure 3.4: SDMX Metamodel

From https://metadatatechnology.com/

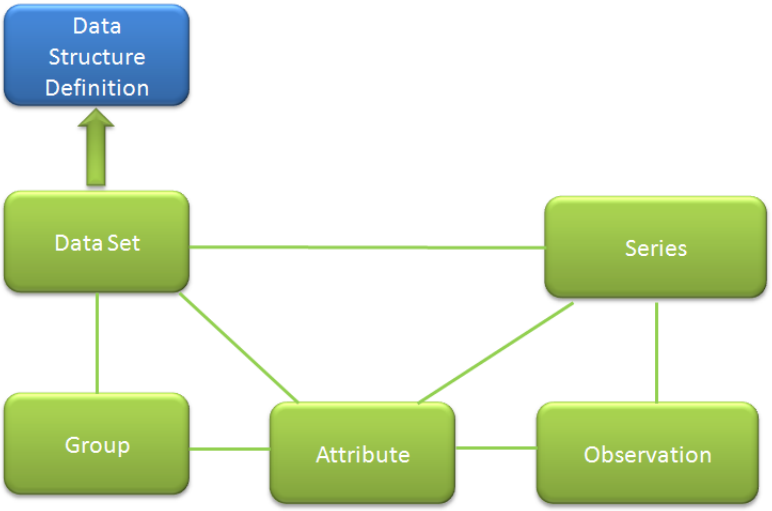

Figure 3.5: Dataset components

From https://metadatatechnology.com/

A dataset can be represented in different formats within the SDMX specification:

1) **Generic Format (fig: 3.6):**

   There is a single XML schema that supports all datasets regardless of the Data Structure Definition (DSD) that defines the allowable content. This format is the most verbose of all the SDMX formats.

```
<generic:Series>
  <generic:SeriesKey>
    <generic:Value id="GEO" value="AT"/>
    <generic:Value id="SEX" value="T"/>
    <generic:Value id="HST" value="NPRV"/>
    <generic:Value id="LMS" value="DIV"/>
    <generic:Value id="CAS" value="POP"/>
    <generic:Value id="POB" value="TOTAL"/>
    <generic:Value id="COC" value="TOTAL"/>
    <generic:Value id="AGE" value="TOTAL"/>
    <generic:Value id="FREQ" value="A"/>
  </generic:SeriesKey>
  <generic:Attributes>
    <generic:Value id="TIME_FORMAT" value="P1Y"/>
  </generic:Attributes>
  <generic:Obs>
    <generic:ObsDimension value="2011"/>
    <generic:ObsValue value="10488.0"/>
    <generic:Attributes>
      <generic:Value id="OBS_STATUS" value="I"/>
      <generic:Value id="OBS_NOTE" value="LMS: Including persons whose same-sex registered partnership was legally dissolved."/>
    </generic:Attributes>
  </generic:Obs>
</generic:Series>
<generic:Series>
  <generic:SeriesKey>
    <generic:Value id="GEO" value="AT"/>
    <generic:Value id="SEX" value="T"/>
    <generic:Value id="HST" value="NPRV"/>
    <generic:Value id="LMS" value="MAR"/>
    <generic:Value id="CAS" value="POP"/>
    <generic:Value id="POB" value="TOTAL"/>
    <generic:Value id="COC" value="TOTAL"/>
    <generic:Value id="AGE" value="TOTAL"/>
    <generic:Value id="FREQ" value="A"/>
  </generic:SeriesKey>
  <generic:Attributes>
    <generic:Value id="TIME_FORMAT" value="P1Y"/>
  </generic:Attributes>
  <generic:Obs>
    <generic:ObsDimension value="2011"/>
    <generic:ObsValue value="17811.0"/>
    <generic:Attributes>
      <generic:Value id="OBS_STATUS" value="I"/>
      <generic:Value id="OBS_NOTE" value="LMS: Including persons in a same-sex registered partnership."/>
    </generic:Attributes>
  </generic:Obs>
</generic:Series>
```

Figure 3.6: Generic Format Dataset

From https://metadatatechnology.com/

2) **Structure Specific (fig: 3.7):**

   Except for the TimeSeries variant of this format any one of or all of the Dimensions can be placed at the level of the observation (thus supporting cross-sectional data) The TimeSeries variant of this format mandates that this is TIME_PERIOD. Multiple measures (specified in the Measure Dimension) can be contained in the Observation. This is the most terse XML format for SDMX data.

```
<Series GEO="AT" SEX="T" HST="NPRV" LMS="DIV" CAS="POP" POB="TOTAL" COC="TOTAL" AGE="TOTAL" FREQ="A" TIME_FORMAT="P1Y">
  <Obs TIME_PERIOD="2011" OBS_VALUE="10488.0" OBS_STATUS="I" OBS_NOTE="LMS: Including persons whose same-sex registered partnership wa
</Series>
<Series GEO="AT" SEX="T" HST="NPRV" LMS="MAR" CAS="POP" POB="TOTAL" COC="TOTAL" AGE="TOTAL" FREQ="A" TIME_FORMAT="P1Y">
  <Obs TIME_PERIOD="2011" OBS_VALUE="17811.0" OBS_STATUS="I" OBS_NOTE="LMS: Including persons in a same-sex registered partnership."/>
</Series>
```

Figure 3.7: Structure Specific Dataset

From https://metadatatechnology.com/

3) **Compact (fig: 3.8):**

Supports Time Series only. This format is similar to the Structure Specific format where TIME_PERIOD is the Dimension associated at the level of the observation: the dataset contains all of the Dimensions except TIME_PERIOD and the TIME_PERIOD is iterated with the observation value at the level.

```
<ns1:Series GEO="AT" SEX="T" HST="NPRV" LMS="DIV" CAS="POP" POB="TOTAL" COC="TOTAL" AGE="TOTAL" FREQ="A" TIME_FORMAT="P1Y">
  <ns1:Obs TIME="2011" OBS_VALUE="10488.0" OBS_STATUS="I" OBS_NOTE="LMS: Including persons whose same-sex registered partnership was leg
</ns1:Series>
<ns1:Series GEO="AT" SEX="T" HST="NPRV" LMS="MAR" CAS="POP" POB="TOTAL" COC="TOTAL" AGE="TOTAL" FREQ="A" TIME_FORMAT="P1Y">
  <ns1:Obs TIME="2011" OBS_VALUE="17811.0" OBS_STATUS="I" OBS_NOTE="LMS: Including persons in a same-sex registered partnership."/>
```

Figure 3.8: Compact Dataset

From https://metadatatechnology.com/

4) **Cross Sectional (fig: 3.9):**

Supports a non-time series representation of the data (though time can be present in the dataset). The placement of the Dimension and Attributes (with the group, series, observation) is specified in the DSD. This can even be a “flat” representation where all Dimensions and Attributes are presented at the same level as the observation value. Multiple measures can be specified in the DSD though this is rather complex, as the DSD must declare explicitly the measures and map each to the relevant code list.

```
<census:Group FREQ="A" TIME="2011" TIME_FORMAT="P1Y">
  <census:Section>
    <census:OBS_VALUE AGE="TOTAL" CAS="POP" COC="TOTAL" GEO="AT" HST="NPRV" LMS="DIV"
    OBS_NOTE="LMS: Including persons whose same-sex registered partnership was legally dissolved." OBS_STATUS="I" POB="TOTAL" SEX="T" value="10488.0"/>
    <census:OBS_VALUE AGE="TOTAL" CAS="POP" COC="TOTAL" GEO="AT" HST="NPRV" LMS="MAR"
    OBS_NOTE="LMS: Including persons in a same-sex registered partnership." OBS_STATUS="I" POB="TOTAL" SEX="T" value="17811.0"/>
```

Figure 3.9: Cross Sectional Dataset

From https://metadatatechnology.com/

The D.I. application uses the SdmxSource [2] java library to interpret the XML messages sent back by the repository regardless of their format and build his own domain model objects using the information described in figures 3.4 and 3.5.

3.2. Data Analytics and Visualization

The Data Analytics and Visualization (D.A.V.) application gives access to the end user to the datasets and services offered by the system. Requirements FR4, FR5 and FR6 from table 2.1 describes what the application must do in order to satisfy the required needs: it must allow Registered Users to visualize the datasets with different types of charts and do different types of analysis with them, make interactive dashboards shareable with also Non-Registered Users and upload their own datasets as CSV or XLS files. Requirement NFR1 from table 2.2 imposes instead that the system must be easy to use also for non-technical users, allowing them to make analysis knowing at most the basis of some data manipulation languages.

Developing from scratch an application that satisfy these needs could be a daunting task, so we have taken into account three different open source projects to be used: Metabase [3], Apache Superset [4] and Knowage [5]. In the following sections we will briefly describe and compare them using the requirements described above, other than their stability and maintenance, as evaluators.

3.2.1. Metabase

Metabase is an open source data visualization and analytics application, build in React and Clojure, with a very intuitive user interface and an API that embeds SQL scripts inside Excel-like functions, giving to the users the ability to write Excel-like formulas to analyze their data. It can be easily integrated with an existing database and ships by default with many drivers already integrated. Although for simple analysis the user does not have to know SQL since all can be done via his user interface, for more complex analysis Metabase offers a good SQL editor with code completion and linting.

From the data visualization point of view, Metabase has quite few options for creating graphs and dashboards, that can enough for simple analysis but not for more complex and advanced projects.

As an overall view, Metabase can be seen as a good data analytics and visualization app but cannot be described as a real Business Intelligence tool since it misses some important and advanced functionalities.

3.2.2. Apache Superset

Apache Superset is a software written in Python, first developed by the AirBnb company and now under the Apache Software Foundation. It has a simple user interface which integrates an interactive widget for simple analysis and a SQL editor for more complex queries and has quite different visualizations options including 3D graphs, heatmaps, pivot tables and so on.

At the time this document has been written, the project is still under development and is not yet on a stable release, so it has not been analyzed further.

3.2.3. Knowage

Formerly known as SpagoBI, Knowage is a Business Intelligence tool written in Java which gives to the user the ability to build complete and advanced dashboard with different types of visualizations that can be highly customized. Knowage has a very clear user role distinction: there are Administrators who can manage everything of the application starting from the other registered users and their roles, to the database connections and datasets creation, then there is the BI developer who has the ability to create the connections between the system and different databases, to create the datasets that will be available to the end users (with the possibility to restrict a specific dataset only to some users with a specific role or authorization) and to create all the types of analytical documents starting from the simple dashboard to more complex analytical documents made using engines like Birt or Olap. Finally, there is the End-User who can access and analyze the datasets shared to him, upload other datasets from CSV or XLS files and share them with other users and create analytical dashboards using the various visualization options.

Differently from the other two solutions, the user interface of Knowage is not very intuitive and requires a bit of knowledge of the environment to make full use of its features. Also, the abstraction from the SQL language is not very high and often is required to write SQL scripts also for simple analysis. The analytical editor is useful for combining different datasets and build very simple indexes but, as we used it, we noticed some functionality bugs that prevent it to work better.

As an overall view, Knowage can be seen as a full Business Intelligence tool with many, also advanced, functionalities. Using Knowage users can build complete dashboards with different types of information and analysis, although it is not very intuitive on the first approach and requires a basic knowledge of SQL language.

3.2.4. Systems comparison

Table 3.1 compares Metabase and Knowage with respect to the requirements defined above, for each requirement is then highlighted in green the application that better satisfies it. Please note that we

excluded Apache Superset from this analysis since it is still in development mode and so we felt that it is not stable enough to be used in the system.

Requirement	Metabase	Knowage
FR4	Although the interface for making analysis is very simple and intuitive, it offers very few options for visualizing the data and very low customization options.	Has different types of visualization options with a high customization and offers the possibility to make advanced kind of analysis using analytical engines like Birt or QBE.
FR5	The dashboards can embed all types of charts and analysis available in the system and can be shared to Non-Registered Users but do not offer any type of interaction (e.g. filtering data, download datasets, etc.).	Allows the creation of dashboards that can be shared to Non-Registered Users and offers different types of interactions, e.g. filtering data, download the displayed data, manipulate graphs, etc.
FR6	Does not offers this functionality.	Allows Registered Users to upload CSV and XLS files and share them with other Registered Users.
NFR1	Easy user interface and high abstraction from SQL for simple analysis.	User interface not very intuitive on the first approach. A base knowledge of SQL is required.

Table 3.1: Business Intelligence systems comparison

As the table shows Knowage satisfies all the Functional Requirements while the Non-Functional Requirements are better satisfied by Metabase. The final decision has been to use Knowage for this first release since the Non-Functional Requirement can be filled with a slight training about the system and the basis of SQL while the lack of functionalities of Metabase would have affected the usefulness of the system.

Figures 3.10 and 3.11 shows two examples of possibles dashboards that can be made using Knowage [2]. Dashboard 3.10 is an example of an informative dashboard showing the result of an analysis made on a dataset collected by the application. In particular this dashboard shows the contribution of the PIL from Abruzzo to the PIL from Italy and South of Italy and its variation over time. Dashboard 3.11 is instead an example of a dashboard for the download of a structured and pre-processed dataset. Registered (and also Non-Registered if the dashboard is public) Users can freely download the data on different formats and can use different filters on specific attributes. This type of dashboard can be very useful to provide users with pre-processed data directly instead of raw data.

[2] The data displayed are Open Data collected from the I.Stat repository (http:\dati.istat.it)

TERRITORI APERTI

Anni

Valore Assoluto

Anno	Abruzzo	Italia	Mezzogiorno
1995	19.426,8	988.243	238.035
1996	20.638,6	1.045.873	251.314
1997	21.312,7	1.092.357	263.768
1998	21.827,8	1.138.856	274.969
1999	22.619,0	1.175.150	284.493
2000	24.134,8	1.241.513	299.192
2001	25.259,9	1.304.137	314.783
2002	25.780,5	1.350.259	324.182
2003	26.221,8	1.394.693	333.480
2004	26.373,4	1.452.319	344.602

1 to 10 di 24 Pagina 1 of 3

Indicizzazione: 1995=100

Anno	Abruzzo	Italia	Mezzogiorno
1995	100	100	100
1996	106	106	106
1997	110	111	111
1998	112	115	116
1999	116	119	120
2000	124	126	126
2001	130	132	132
2002	133	137	136
2003	135	141	140
2004	136	147	145

1 to 10 di 24 Pagina 1 of 3

Quote

Anno	Abr/It	Abr/Mez
1995	1,97	8,16
1996	1,97	8,21
1997	1,95	8,08
1998	1,92	7,94
1999	1,92	7,95
2000	1,94	8,07
2001	1,94	8,02
2002	1,91	7,95
2003	1,88	7,86
2004	1,82	7,65

to 10 di 24 Pagina 1 of 3

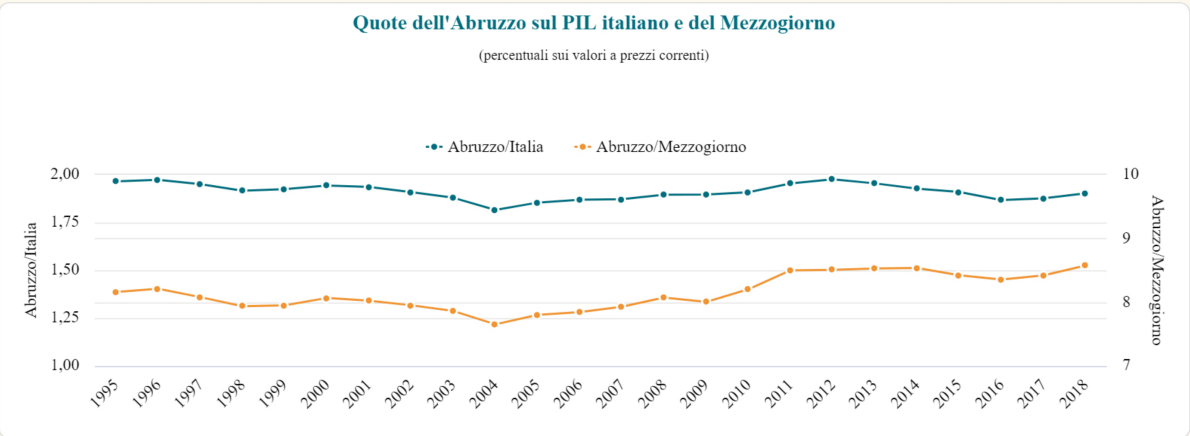

Figure 3.10: Example of informative dashboard

Territorio	Edizione	Periodo
Italia	Mag-2020	2018, 2017, 2016

ANNO 2018, 2017, 2016 TERRITORIO Italia EDIZIONE Mag-2020

Sequenza dei conti

			2016	2017
conto_della_generazione_dei_redditi_primari	conto_della_generazione_dei_redditi_primari	prodotto_interno_lordo_ai_prezzi_di_mercato	1.695.786,80	1.736.592,80 1
		risultato_lordo_di_gestione_e_reddito_misto_lordo	818.222,50	830.901,40
	impieghi	imposte_indirette_nette	210.342,00	221.638,00
		redditi_interni_da_lavoro_dipendente	667.222,30	684.053,40
conto_economico_delle_risorse_e_degli_impieghi	impieghi	consumi_finali_interni	1.360.939,20	1.392.857,70
		investimenti_fissi_lordi	291.183,50	303.569,90
		variazione_delle_scorte_e_acquisizioni_meno_cessioni_di_oggetti_di_valore	6.614,00	9.955,80
	risorse	importazioni_nette	-37.050,00	-30.209,40
		prodotto_interno_lordo_ai_prezzi_di_mercato	1.695.786,80	1.736.592,80 1

Figure 3.11: Example of dashboard for the download of a dataset

4 Conclusions and future works

This deliverable described the basic concepts, the main architecture and the first implementation of the TA-Analytics system.

In its first stage we have defined the early architecture of the system in order to provide a core basic set of service. A lot of effort has been given to the development of the Data Importer application and to the processing of the SDMX messages due to their particular structure which makes very difficult to parse the XML messages using a standard XML schema. The existence of a specific library for the parsing of SDMX messages [2] helped us to solve this task.

The choice of a good Business Intelligence application as D.A.V. component was a tricky task as well, since we needed to take account of the various Functional Requirements as well as the ease of use the entire system. The availability of open source B.I. applications is wide and so we first had to filter the ones satisfying most of the needed requirements, leading to the applications described in section 3.2. After this first selection we then had to carefully analyze the functionalities of the selected applications in order to find the one that best satisfies our requirements, as said in section 3.2.4.

Finally, the integration between the D.I. and D.A.V. components has been not so hard since they share the same database (section 2.3) and so all the process has been reduced to connecting these two components to a common database to give access to the same datasets.

The area of improvements for the next releases are manifold both for the D.I. and D.A.V. components: about the first component one point is for sure the integration of more open repositories (and correspondents web services) to the system as well as the improvement of the performances for processing expecially large SDMX datasets. About the D.A.V. component instead future works will be the improvement of the user interface as well as his easy of use and abstraction from the data manipulation languages used and also the integration of the functionality of adding machine learning scripts and custom charts to dashboards to enrich the overall analysis their expressive power.

Bibliography

[1] “Spring batch.” [Online]. Available: https://spring.io/projects/spring-batch

[2] K. Nicholls, “Sdmxsource: Metadata technology.” [Online]. Available: https://metadatatechnology.com/software/sdmxsource.php

[3] “Metabase.” [Online]. Available: https://www.metabase.com/

[4] “Apache superset (incubating).” [Online]. Available: https://superset.incubator.apache.org/

[5] “Open source business intelligence - knowage suite.” [Online]. Available: https://www.knowage-suite.com/site/