WP3 Extracting city knowledge for intelligent services
D3.2 Big Data Analytics Framework report – first release

Grant Agreement N°723139
NICT management number: 18301

BIGCLOUT

Big data meeting Cloud and IoT
for empowering the citizen ClouT in smart cities

H2020-EUJ-2016 EU-Japan Joint Call

EU Editor: **ICCS** JP Editor**: TSU**

Nature: Report

Dissemination: PU

Contractual delivery date: 2018-07-01

Submission Date: 2018-07-09

Co-funded by the EU H2020 GA. 723139 and NICT GA. 18301

ABSTRACT

This deliverable consists in a technical report of the first version of Big Data Analytics Framework of BigClouT that is identified in the general BigClouT architecture and investigates in depth the functional subcomponents of the City Data Processing in order to enable their future integration and implementation through various use case scenarios.

Disclaimer

REVISION HISTORY

Revision	Date	Description	Author (Organisation)
V0.1	26.03.2018	Initial ToC	Antonis Litke (ICCS)
V0.2	08.05.2018	Final ToC	Orfeas Voutyras (ICCS)
V0.3	25.05.2018	Sections 2 & 4: Initial Input	Guiseppe Ciulla (ENG)
V0.4	28.05.2018	Section 3	Savong Bou, Toshiyuki Amagasa, and Hiroyuki Kitagawa (TSU)
V0.5	01.06.2018	Section 5	George Palaiokrassas (ICCS)
V0.6	08.06.2018	Sections 2 & 4: Update	Guiseppe Ciulla (ENG)
V0.7	19.06.2018	Section 4.3 StreamOLAP	Savong Bou, Toshiyuki Amagasa, and Hiroyuki Kitagawa (TSU)
V0.8	26.06.2018	Sections 2 & 4: Final	Guiseppe Ciulla (ENG)
V0.9	28.06.2018	Section 5: Final	George Palaiokrassas (ICCS)
V0.10	30.06.2018	Version ready for internal review	Orfeas Voutyras (ICCS)
V0.11	02.07.2018	Section 4.4 DeepOnEdge	Takuro Yonezawa (KEIO)
V0.11	03.07.2018	Internal Review	Guiseppe Ciulla (ENG)
V0.12	04.07.2018	Internal Review	Sato Hiroshi (NTTRD)
V1.0	06.07.2018	Version ready for submission	Orfeas Voutyras (ICCS)

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACRONYM	DEFINITION
API	Application Programming Interface
CNN	Convolutional Neural Network
CSaaS	City Software as a Service
DoE	Deep on Edge
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HA	High Availability
HTTP	Hypertext Transfer Protocol
ILSVR	ImageNet Large Scale Visual Recognition
IT	Information Technology
LOV	List Of Values
ML	Machine Learning
OLAP	OnLine Analytical Processing
QbE	Query by Example
REST	REpresentational State Transfer
RDBMS	Relational Database Management System
SNS	Social Networking Service
SPE	Stream Processing Engine
SVM	Support Vector Machine classifier
URL	Uniform Resource Locator

Executive Summary

This deliverable consists in a technical report of the first version of Big Data Analytics Framework of BigClouT that is identified in the general BigClouT architecture and investigates in depth the functional subcomponents of the City Data Processing in order to enable their future integration and implementation through various use case scenarios.

The document aims at presenting in depth the technical details of each and every functionality offered by the several assets that have been adopted or created under the framework of WP3. For this purpose, we take a modular approach where the functionalities of city data processing module are presented individually. This categorisation is aligned to the tasks of WP3 and focuses on the extraction of city knowledge for intelligent services by providing a common framework.

The goal behind following a detailed presentation of the elementary services offered by the several assets provided by the partners of the consortium is twofold:

- Facilitating the collaboration between technical partners of the project (from both WP2 and WP3), thus supporting the future **integration** plans and actions between the various components already provided.
- Facilitating the discussions between the pilot partners and technical partners, thus enabling the future **demonstration** of the several services provided by the project through various use case scenarios.

To this end, Section 2 presents the updated WP3 architecture. Sections 3, 4, 5 and 6 describe in detail all the WP3 elementary services in a manner that supports the two aforementioned goals (more details in Section 1.3). Section 7 briefly presents the integration plan, which will be used as a roadmap for the final year of the project. Finally, Section 8 concludes this report.

1 Introduction

1.1 Objectives and goals of the task and deliverable

This deliverable (D3.2) aims at presenting in depth the technical details of each and every functionality offered by the several assets that have been adopted or created under the framework of WP3. For this purpose, we take a modular approach where the functionalities of city data processing module are presented individually.

The functionalities are grouped under four different categories, namely:

1) Data Event Processing,
2) Big Data Analysis,
3) Machine Learning, Predictive Modelling and Decision making,
4) and Visualisation

This categorisation, which is aligned to the tasks of WP3, focuses on the extraction of city knowledge for intelligent services by providing a common framework:

- T3.1 Big data analytics and business intelligence
- T3.2 Learning , predictive modelling and decision making
- T3.3 Distributed real-time data mining with event detection for actuation

The goal behind following a detailed presentation of the elementary services offered by the several assets provided by the partners of the consortium is twofold:

- Facilitating the collaboration between technical partners of the project (from both WP2 and WP3), thus supporting the future **integration** plans and actions between the various components already provided.
- Facilitating the discussions between the pilot partners and technical partners, thus enabling the future **demonstration** of the several services provided by the project through various use case scenarios.

To this end, Section 2 presents the updated WP3 architecture. Sections 3, 4, 5 and 6 describe in detail all the WP3 elementary services in a manner that supports the two aforementioned goals (more details in Section 1.3). Section 7 briefly presents the integration plan, which will be used as a roadmap for the final year of the project. Finally, Section 8 concludes this report.

1.2 Relation to other WPs and Tasks

This document is building on top the work presented in "**D3.1 Big Data Analytics Framework Architecture**" during the first year of the project. Updating the general WP3 architecture, providing more details and new technical details of the several components and focusing on the future integration and demonstration of the WP3 services are three of the main differences between the two deliverables.

While technical details, such as functional description and implementation details are provided in D3.2, "**D3.3 Big Data Analytics Framework Prototype – Demonstration**" focuses on deployment and usage of each functionality. In that sense, the two deliverables are complementary to each other. It should be noted that Sections 3, 4, 5 and 6 of this document are mapped one-to-one with Sections 2, 3, 4 and 5 of D3.3, since they present the same functionalities.

Finally, this deliverable is closely related to "**D4.3 Integrated use cases and first large-scale deployments and experimentation**". As it will be made clear in Section 1.3, most of the subsections of Sections 3, 4, 5 and 6 are structured in such a manner that they can provide crucial input regarding future integration and demonstration plans.

1.3 Methodology followed

Due to the wide range of services that can be provided under the four main modules of the WP (Data Event Processing, Big Data Analysis, Machine Learning, Predictive Modelling and Decision making, and Visualisation), each one of them (presented in Sections 3, 4, 5 and 6) may include more than one services provided by the current assets of the project.

Likewise, due to the wide range of functionalities of the assets and the modular approach we follow, an asset may appear in more than two categories/modules (like in the case of KNOWAGE in Sections 4 and 6).

For this deliverable to act as a reference document and a roadmap for future plans and actions regarding integration and (integrated) demonstrations, the elementary services presented in Sections 3, 4, 5 and 6 adopt the following structure:

- **Description of Functionalities:**
  Giving a general description of the main functionalities of the service/tool is given, as a short introduction to its capabilities.

- **Implementation details and Internal Architecture:**
  Showing how the above functionalities, probably represented by specific [sub]components, are connected with each other and relate to each other.

- **Interfaces and Integration:**
  Listing the UIs, APIs, etc. that can be used to give input to the module and access its output, thus presenting the possibilities of interaction with both end-users and other components (in WP3/4).

- **Performance, Evaluation and Stress-tests:**
  Presenting more technical characteristics, showing how the module performs when it comes to requirements as presented in past deliverables (big data, granularity of data, response time, number of nodes that can be managed at the same time, integration, etc.).

- **Candidate Use Cases to be supported:**
  Suggesting candidate Use Cases that could be supported by the component, given the functionalities of the module and the requirements/description of the Use Cases of the project presented in D1.4. It should be stressed out that not all of the presented use case scenarios will be implemented: the scenarios presented aim at creating some first common ground between WP3 and WP4.

2 Updated WP3 architecture

This section provides information and details about the final version of the sub architecture of BigClouT City Data Processing module that represents the Big Data Analytics Framework. Definition of the final version of this sub architecture starts from the final architecture of BigClouT reported in "D1.4 Updated use cases, requirements and architecture" and from its initial version reported in "D3.1 Big Data Analytics Framework Architecture".

Indeed, coherently with the general approach of BigClouT architecture, this sub architecture must be considered as a reference architecture characterised by flexibility and adaptability that can be adapted and customised to address specific needs and requirements.

The aim of the final version of this sub architecture is to consolidate the logical organisation of its macro modules and of the technological assets.

As reported in D3.1, the Big Data Analytics Framework matches with the logical module "City Data Processing" of the general logical BigClouT architecture (Figure 1) and its subcomponents, except for the subcomponent "Context Management & Self-Awareness" that is investigated in WP2.

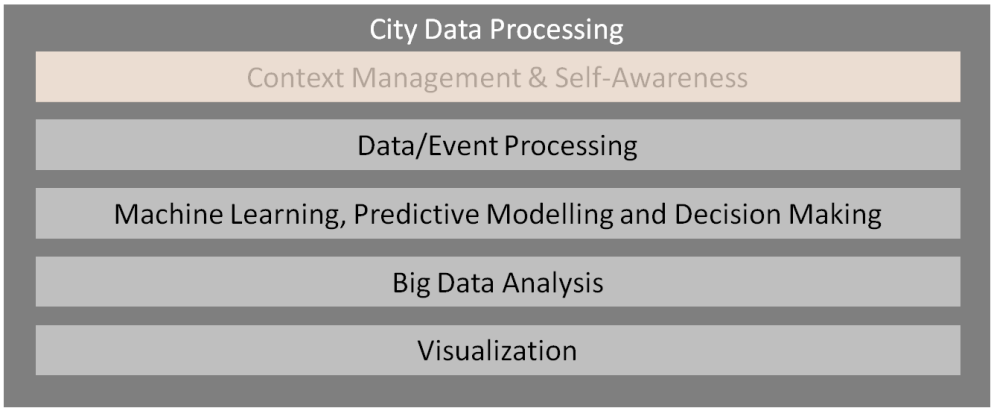

FIGURE 1: CITY DATA PROCESSING

Logical relations among these subcomponents and assets mapping are reported in Figure 2. As depicted, some assets are reported in more than one logical subcomponents. This is due to the fact that these assets provide different functionalities that fulfil different aims. For instance, CoherentPaaS provides useful functionalities for both "Machine Learning, Predictive Modelling and Decision Making" and "Data/Event Processing" subcomponents.

In Figure 2, lines of different colours are used only to avoid confusion in the relations depicted in the figure.

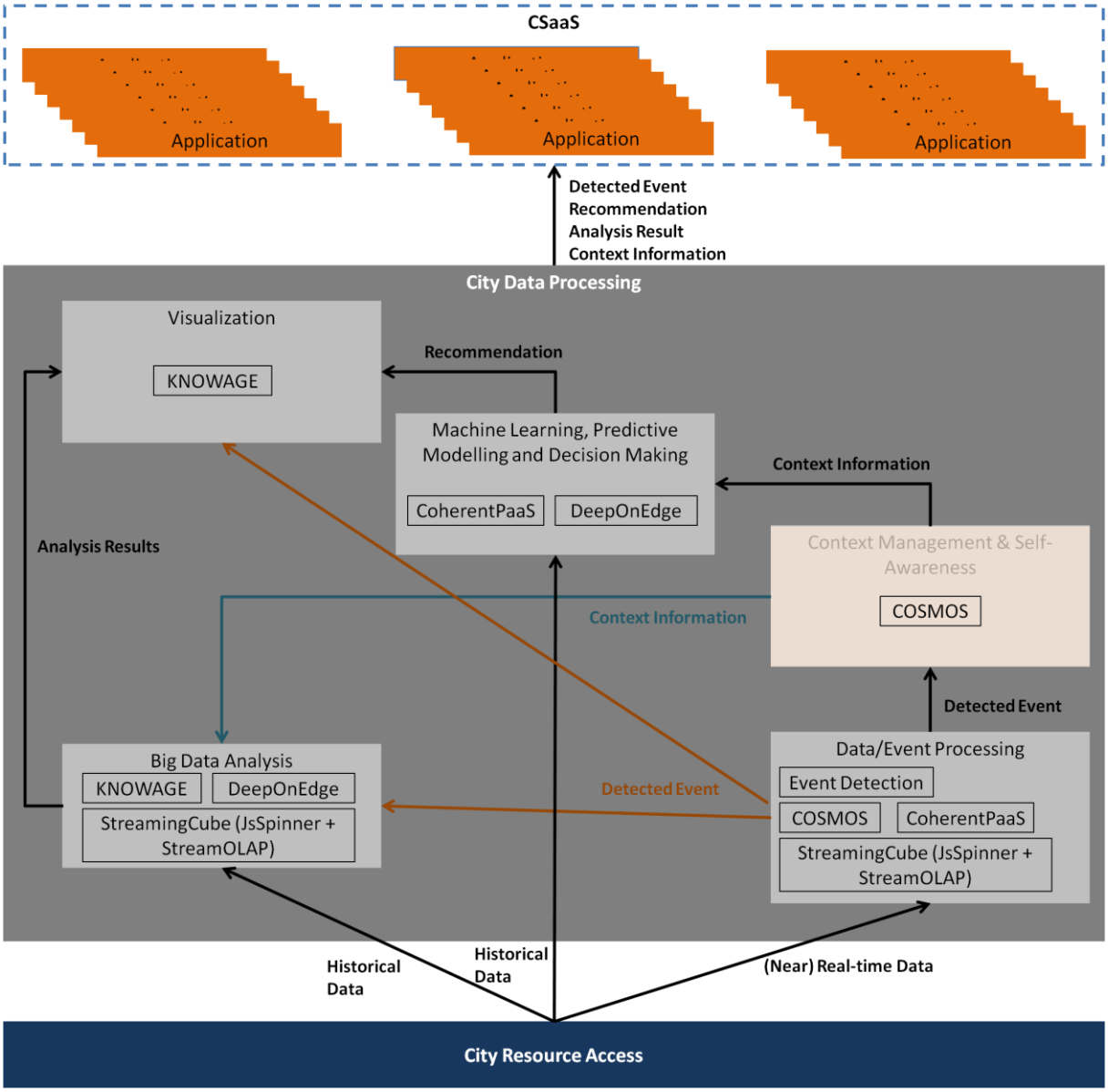

FIGURE 2: RELATIONS BETWEEN SUBCOMPONENTS OF CITY DATA PROCESSING AND ASSETS MAPPING

3 Data Event Processing

3.1 General Description

Data Event Processing aims at accepting and manipulating city data from the City Resource Access module in usable and desired form. It also provides the data to any other component of the City Data Processing requesting for further analysis from it to get more insight information. Accessing and accepting data from the City Resource Access module can be done either in a synchronous way (client/server mode) or in an asynchronous way (publish/subscribe mode). Finally, it also provides a way to interact with actuators (actuating), and to push data to a connected counterpart.

3.2 JsSpinner

3.2.1 Description of Functionalities

City data are initially in the form of event transactions or some observations. The Data Event Processing component is in charge of collecting and manipulating the data by converting them into a usable and desired form. The manipulation involves some operations, such as filtering, joining, calculating, aggregating, etc. The four main functionalities provided by Data Event Processing are explained as follows:

Functionality 1 – Data Filtering

This functionality is responsible for refining city data into what users need, without including other data that can be repetitive, irrelevant, or even sensitive.

Functionality 2 – Data Joining

This function systematically combines city data from different sources into a more integrated piece of information. This functionality is very important in making city data processing smooth and efficient.

Functionality 3 – Event Detecting

The aim of this functionality is to process city data in order to detect events of interest providing capabilities for both event processing and stream processing. Input data of this subcomponents will be data coming from sensors, web pages, etc., provided by the City Resource Access module. Outputs are the detected events.

Functionality 4 – Data Aggregating

The information is gathered, grouped, and summarized to obtain abstract or higher-levels of knowledge. Raw data is aggregated over a given time period or some specified properties to provide statistics such as average, minimum, maximum, sum, count, etc.

3.2.2 Implementation details and Internal Architecture

3.2.2.1 Internal Architecture

The Data Event Processing component with respect to BigClouT architecture is shown in Figure 3. It belongs to the City Data Processing Module. Its internal architecture is shown in Figure 4. As shown in the figure, Data Event Processing directly accepts either historical data or (near) real-time data from the City Resource Access Module of the BigClouT through I/O manager. Then, the accepted data are manipulated by applying various functions for different purposes. Four main manipulating functions provided by this component are data filtering, data joining, event detecting, and data/event aggregating. The final products of Data Event Processing are filtered data, joint data, detected events, and aggregated data, which are sent to other components of the City Data Processing Module for further analysis to get more useful information. I/O manager is also responsible for dispatching the outputs of Data Event Processing to other components of the City Data Processing Module.

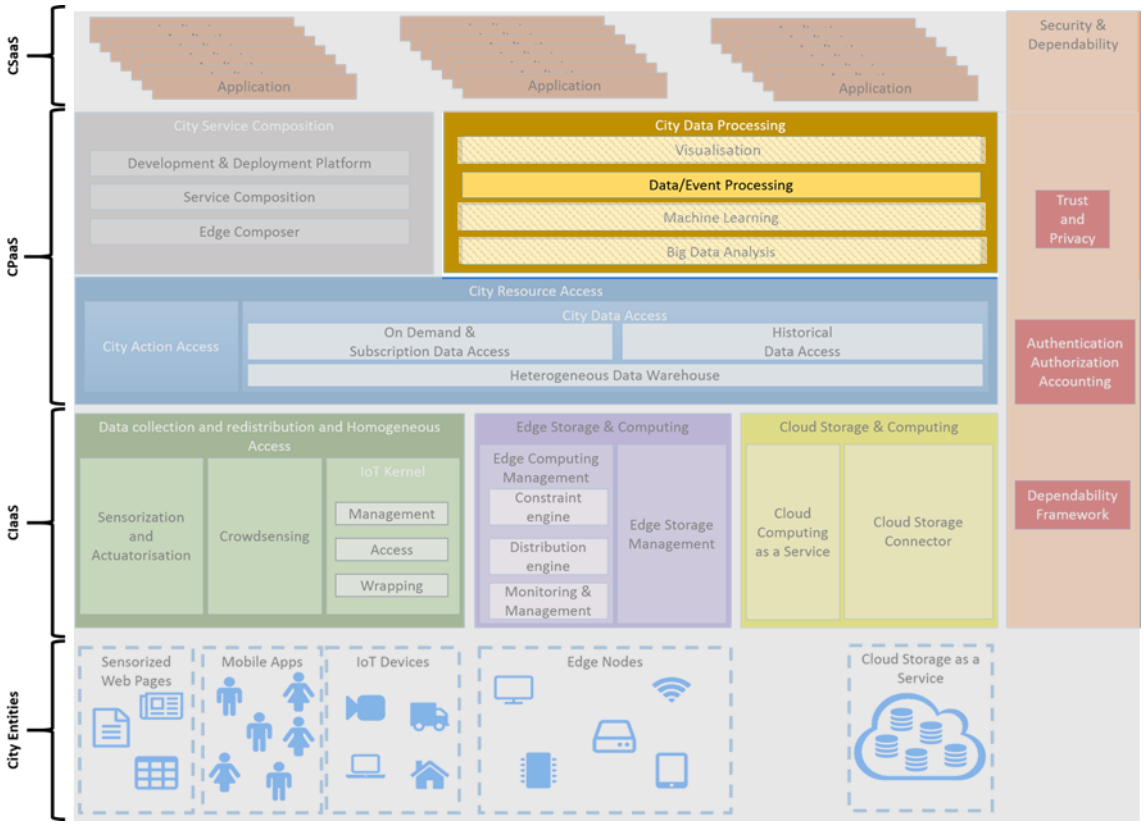

FIGURE 3: DATA EVENT PROCESSING IN BIGCLOUT ARCHITECTURE

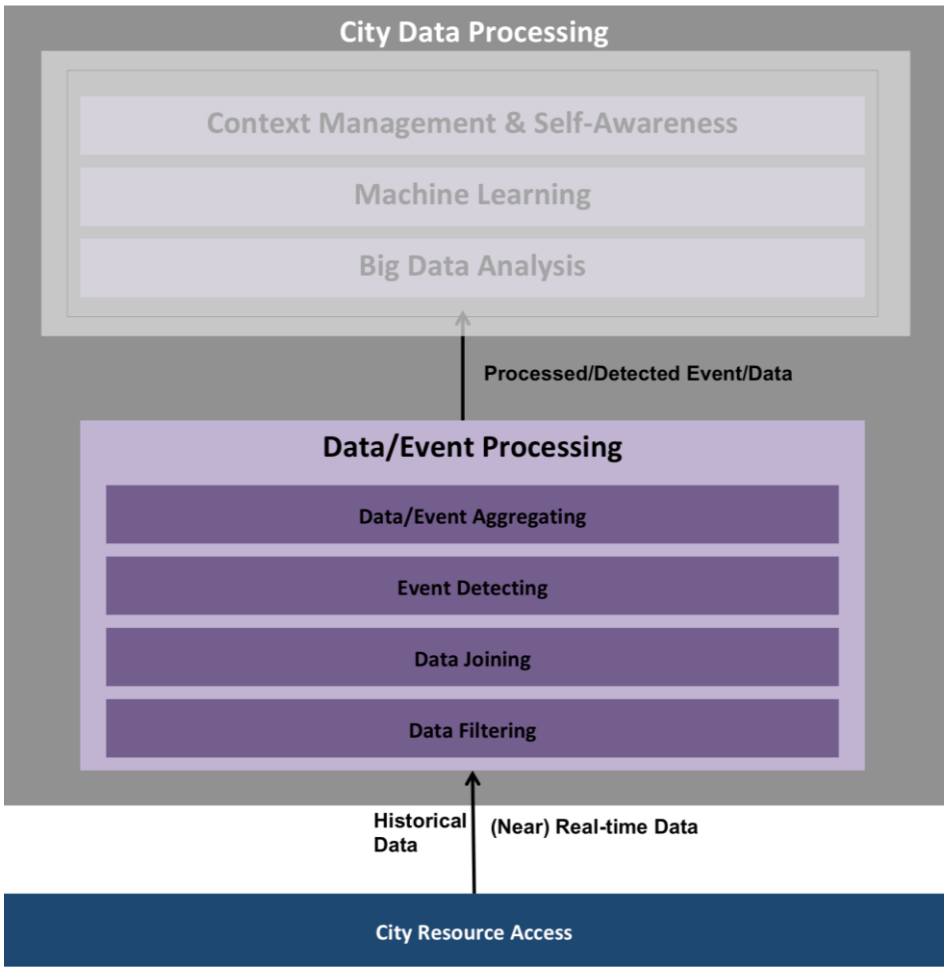

FIGURE 4 INTERNAL ARCHITECTURE OF DATA EVENT PROCESSING

3.2.2.2 Relationship between Each Functionality

All functionalities of the Data Event Processing are interrelated to each other in terms of dispatching processing data back and forth, which is depicted in Figure 5, Figure 6, Figure 7 and Figure 8.

The sequence diagram in Figure 5 describes the relations between functionalities of Data Event Processing with respect to the filtered data they exchange.

The flow is initiated by the City Resource Access; it provides the Data Filtering subcomponent with the historical or real-time or near real-time data coming from sensors, web pages, crowd-sensing, etc. connected to the BigClouT platform.

When new data arrive, the Data Filtering regularly takes out unneeded information that does not match the filtering conditions. Then, the filtered data is dispatched to the other subcomponents of the Data Event Processing.

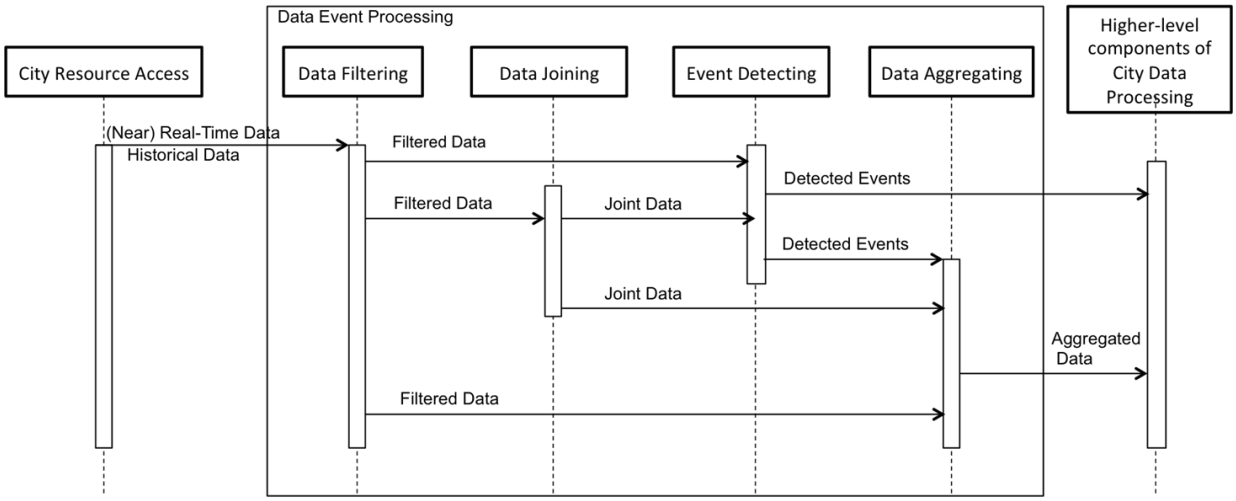

FIGURE 5- EVENT DATA PROCESSING DATA FILTERING SEQUENCE DIAGRAM

The sequence diagram in Figure 6 describes the relations between functionalities of Data Event Processing with respect to the joint data they exchange.

The flow is initiated by the City Resource Access. When new data arrives, the Data Joining regularly combines different city data into an integrated compact piece of information. Then, the joint data is dispatched to the Event Detecting or Data Aggregating subcomponents of the Data Event Processing.

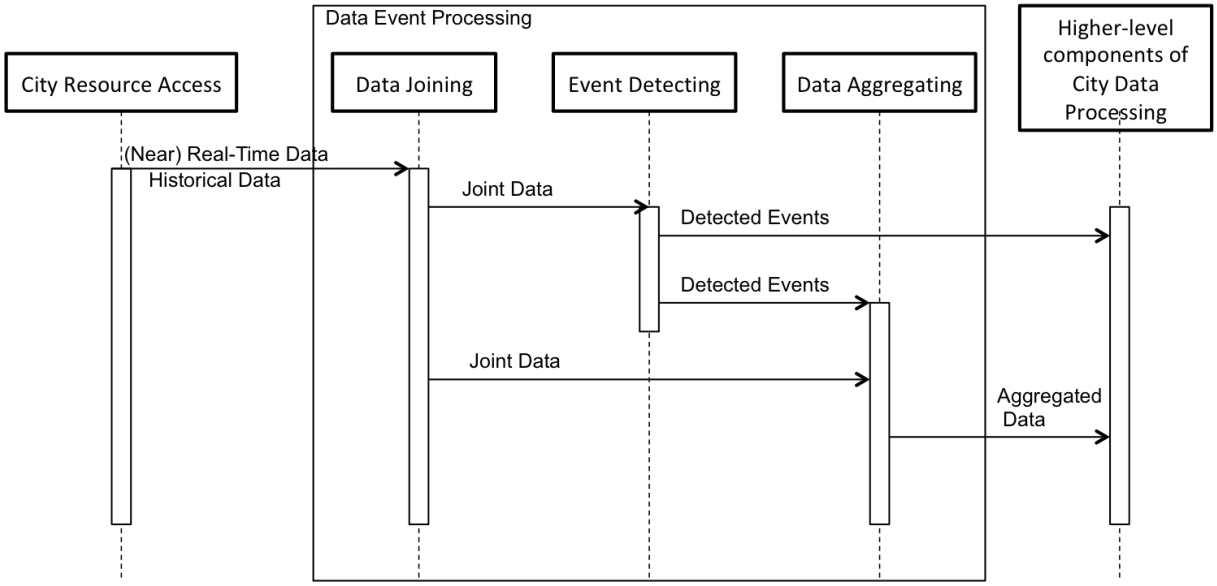

FIGURE 6 EVENT DATA PROCESSING DATA JOINING SEQUENCE DIAGRAM

The sequence diagram in Figure 7 describes the relations between functionalities of Data Event Processing with respect to the detected events they exchange.

The flow is initiated by the City Resource Access. When new data arrive, the Event Detecting regularly executes a process to check if incoming data match with events of interest; when an event of interest is detected (represented in the sequence diagram with the label "Detected Event"), and the data are dispatched to the other logical subcomponents of the City Data Processing or to higher-level components of the City Data Processing Module.

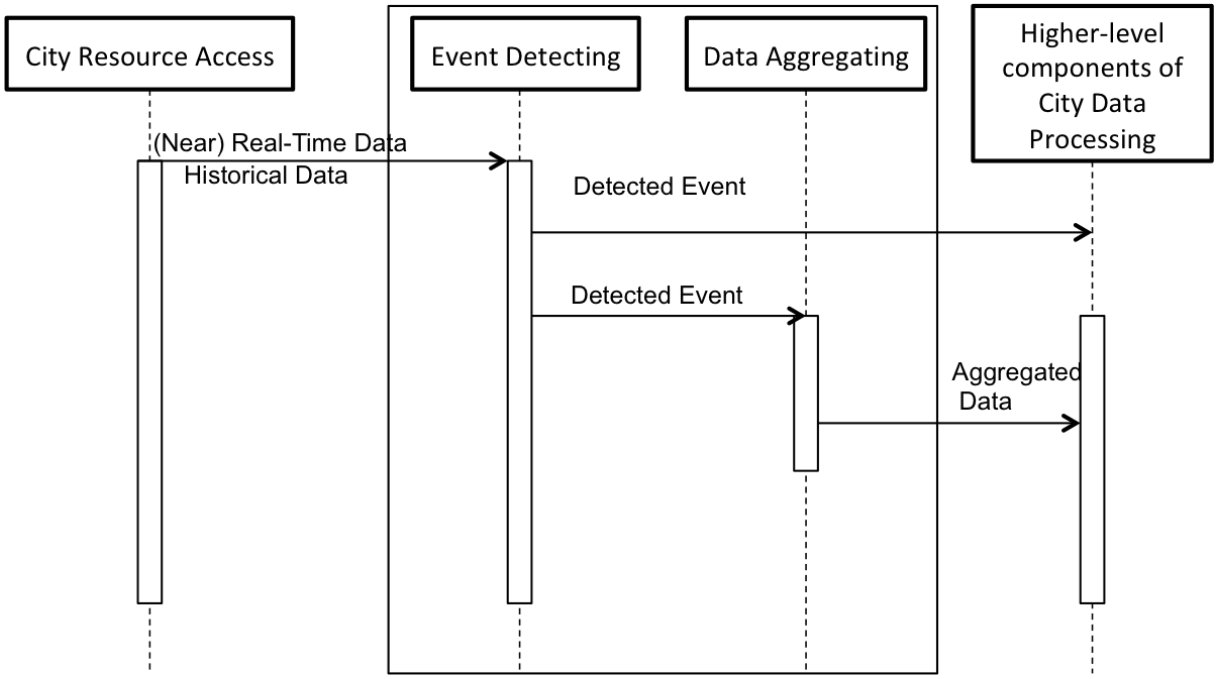

FIGURE 7 EVENT DATA PROCESSING EVENT DETECTING SEQUENCE DIAGRAM

The sequence diagram in Figure 8 depicts the relations between functionalities of Data Event Processing with other higher-level components of City Data Processing in terms of the aggregated data they exchange.

The flow is initiated by the City Resource Access. When new data arrive, the Data Aggregating groups and summarises the obtained data to get higher-level of knowledge by applying various aggregating operations, such as average, minimum, maximum, sum, etc. The aggregated results are dispatched to the other higher-level logical subcomponents of the City Data Processing, in particular to Big Data Analysis and to Context Management & Self-Awareness.

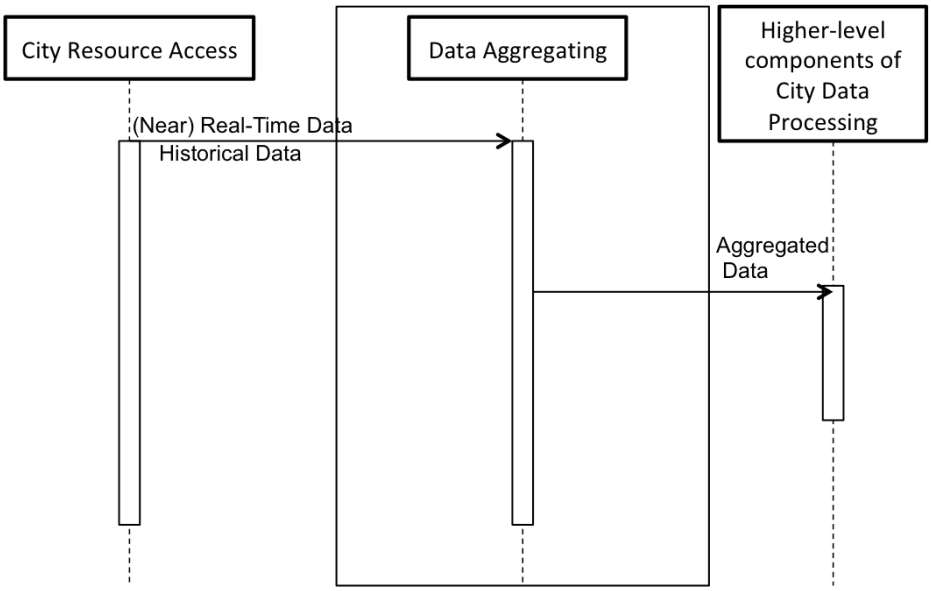

FIGURE 8 EVENT DATA PROCESSING DATA AGGREGATING SEQUENCE DIAGRAM

3.2.3 Interfaces and Integration

3.2.4 Integration

I/O manager of the Data Event Processing component is responsible for dealing with and accepting data streams from the City Resource Access module (Figure 4). City data from the City Resource Access module is accessible by either HTTP request or REST API. Thanks to the well-defined I/O manager, Data Event Processing can connect with City Resource Access module with minimum customisation to the I/O manager by either:

1. **Direct Connector**: Directly add functions to the I/O manager of the Data Event Processing to read/access city data from the City Resource Access module.
2. **Socket Stream Connector**: The City Resource Access module directly sends the city data to the Data Event Processing component.

Both approaches can be done either in a synchronous way (client/server mode) or in an asynchronous way (publish/subscribe mode) by requesting or subscribing for city data from the City Resource Access module.

Notice that, the above methods require the registration of schema of the city data to the I/O manager of Data Event Processing. The schema is in JSON format textual document. Figure 9 shows a sample schema of city data.

FIGURE 9 SAMPLE SCHEMA OF CITY DATA

In addition, I/O manager is also responsible for dispatching the output of Data Event Processing to other logical subcomponents of the City Data Processing.

3.2.5 *Input and Output Specifications*

The Data Event Processing module accepts any data streams of JSON format from the City Resource Access module. The output is in JSON document format.

For example, a sample element is shown in Figure 10 below. For this sample element, the attributes are: prodID, suppID, promoID, custID, custName, storeID, storeArea, and salesAmountFact.


```
{
   "prodID":10001,
   "suppID":20001,
   "promoID":30001,
   "custID":40001,
   "custName":"University of Tsukuba",
   "storeID":50001,
   "storeArea":"Ibaraki Prefecture",
   "salesAmountFact":95
}
```


FIGURE 10 SAMPLE INPUT ELEMENT

3.2.6 *Performance, Evaluation and Stress-tests*

Data Event Processing can efficiently process large-scale city data. Stress-tests were performed and the results were shown in Figure 11. The tested queries involve joining two different streams. As it can be seen, Data Event Processing, whose two execution modes are shown in the figure, can efficiently process data streams with more than 100k tuples/s, which is good enough for real-life big data processing for BigClouT (as presented in D1.4 "Updated Use Case Requirements and Architecture).

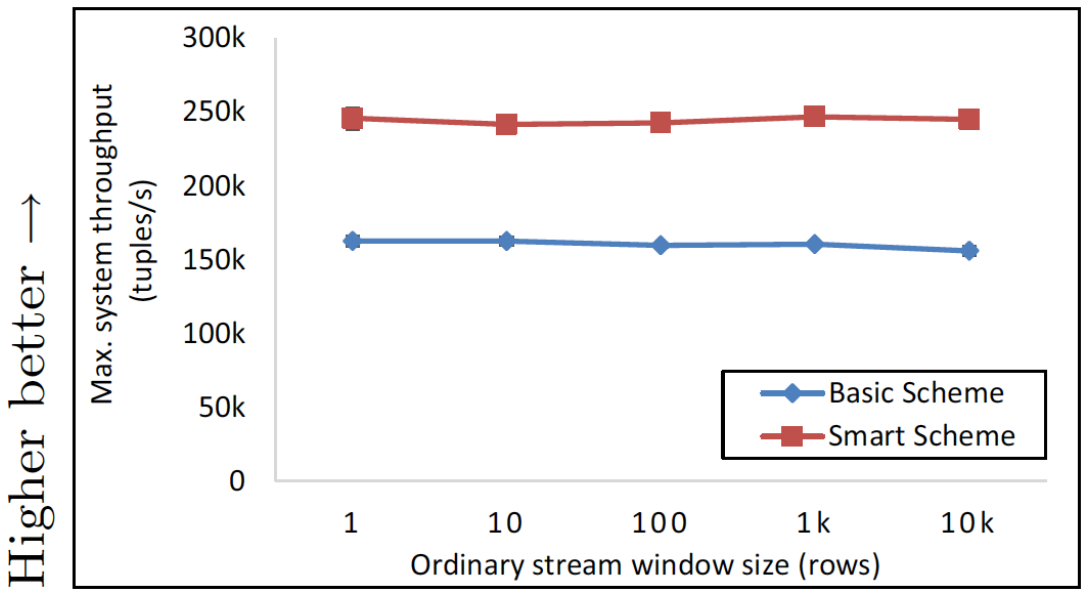

FIGURE 11 MAXIMUM SYSTEM THROUGHPUT EVALUATION

3.2.7 *Candidate Use Cases to be supported*

Data Event Processing functionalities are very indispensable to the BigClouT as well as all its use cases. Filtering, Joining, Detecting and Aggregating functionalities can manipulate city data by converting it into usable and desired form. Table 1 shows in detail which requirements of all use cases are supported by the Event Data Processing Component.

TABLE 1: USE CASES REQUIREMENTS SUPPORTED BY EVENT DATA PROCESSING

Use Case	Requirement Description
Grenoble Use Case 1: Monitoring of economic impacts of events	The application should provide a tool to analyse data and extract statistics in simple and easily understandable way for the city economic development division and for the event organisers.
	The event organisers may share some information about the participant collected during his/her registration to the event (hotel stayed, personal profile, etc.).
Grenoble Use Case 2: Monitoring of Industrial Estates	The Innovallée association should be able to collect (in real-time when relevant) information from the associated stakeholders (restaurants, shops, company associations, etc.) and provide it to the BigClouT platform
	The Grenoble mobility service should provide the information about the transportation system in Grenoble (bus/tram stations, timetables, real-time traffic information, car-sharing, etc.).
	The application should provide a tool to analyse data and extract statistics in simple and easily understandable way for the city economic development division and for Innovallée Association.
	The application should notify the user when interesting events occur and/or customised recommendations to be provided.
Bristol Use Case 1: Smart Energy - predictive analysis of users' power consumption	The BigClouT platform should be able to collect information from sensors deployed inside citizen homes.
	The BigClouT platform should be able to collect detailed weather information in a real time and in an accurate way.
	The BigClouT platform should be able to provide tools for comparing citizens' energy consumption scenarios.
Bristol Use Case 2: Smart Mobility	The BigClouT platform should be able to collect measurements of air quality from sensors deployed around University of Bristol
	The BigClouT platform should be able to analyse citizens' mobility patterns and use data analytics for the prediction of future citizens' mobility patterns.
	The BigClouT platform should be able to merge information with other available sources (i.e. weather information).
Tsukuba Use Case 1: Provide tourism, traffic and environmental information in real time to visitors	A method, for instance SNS (Social Networking Service) analysis, questionnaires by smart phone or interactive signage, should be provided for collecting information to analyse visitors' trends: visitors' attributes, visitors' behaviour history, location information, satisfaction rate for visitors' purpose and their needs, environmental information, statistics from local government.
	The application should be able to provide the most suitable information in real time such as weather information, SNS

	comments, traffic and facilities congestion information by their attribute and location information.
	The application should be able to analyse the trend of satisfaction rate, weather information and visitors' behaviour by their attribute.
Tsukuba Use Case 2: Grasp status about foreign visitors to Tsukuba and provide concierge service to them	A method, (e.g.: SNS analysis, questionnaires by smart phone or interactive signage) should be provided for collecting information to analyse foreign visitors' trends: visitors' attributes, visitors' behaviour history, location information, satisfaction rate for visitors' purpose and their needs, environmental information, statistics from local government, foreign visitors' needs and problems.
	The application should be able to collect accumulated information to provide the most suitable information.
	The application should be able to prepare to answer to solve their needs and problems by collected data according to visitors' behaviour and their attribute.
Fujisawa Use Case 1: Optimising the incidence on local economy of Fujisawa	Various city stakeholders (from IT or city tourism division) should be able to monitor these incident reporting.
	The BigClouT platform should be able to collect real-time city data from various information resources with low cost.
	The application should be able to analyse real-time data efficiently and effectively.
Fujisawa Use Case 2: Fine-grained city infrastructure management	City stakeholders should be able to collect data from sensors attached to garbage collection trucks.
	The BigClouT platform should provide a method to collect real-time city data from various information resources with low cost.
	The BigClouT platform should provide edge-side computing technology to reduce data size with finding important aspect of the data.
	The BigClouT platform should provide a method to share collected data among various stakeholders.
	The BigClouT platform should provide a method to analyse real-time data efficiently and effectively.

4 Big Data Analysis

4.1 General Description

The value creation obtained from the use of Big Data represents an important instrument in analytics and decision-making processes in different fields (e.g.: business, sciences, society, etc.) that can bring a big improvement in city life at different levels and for different actors (e.g.: citizens, municipality, companies, etc.). Making sense and value of big data is a challenge of the BigClouT project that involves the resolution of some problems, such as the ability to analyse data from heterogeneous sources and latency of processing data. BigClouT addresses this type of challenge aiming to a set of business intelligence tools able to extract and analyse these heterogeneous data sources and datasets. The following sections describe how the technological assets belonging to *Big Data Analysis* block (Figure 1) contribute to address these challenges.

In particular, these technological assets are:

- KNOWAGE, that provides capabilities to access data coming from different and heterogeneous data sources and to perform analysis on them.
- StreamingCube, that comes from the union of *JsSpinner* and *StreamOLAP* and provides capabilities for analysis of stream of data.
- DeepOnEdge, that provides specific capabilities for data analysis in edge devices.

4.2 KNOWAGE

KNOWAGE is a suite for Business Intelligence and provides various functionalities; the aim of this section is to report and introduce mainly the functionalities related to Big Data analysis; in particular KNOWAGE gives the chance to users to interact with Big Data sources in order to perform analysis, such as data extraction and correlation. KNOWAGE is available in two versions: the Community Edition (KNOWAGE CE) and the Enterprise Edition (KNOWAGE EE); KNOWAGE CE is freely available, whereas KNOWAGE EE is available only by subscription. Both KNOWAGE CE and KNOWAGE EE provide Big Data analytical features. This section of the document, as well as Section 6.2, refers to KNOWAGE CE version.

4.2.1 Description of Functionalities

In order to perform analysis, KNOWAGE[1] mainly relies on the concepts of Data sources and Datasets. A Data Source is a DB connection used by KNOWAGE to retrieve datasets. A Dataset is the portion of data coming from one or more data sources, or from external data providers that will be used to perform analysis. This section will cover the operation to be performed in order to create Data sources, Datasets and the filters used to guide analysis. Section 6.2.1 will cover the definition of analytical documents and visualisations.

Data Source Definition

A Data Source, as previously defined, is a DB connection. KNOWAGE supports two different connection options:

- JDBC connections, managed directly by KNOWAGE;

[1] https://www.knowage-suite.com/site/home/

- connections retrieved as JNDI, which are managed by the application server (e.g. Tomcat) on which KNOWAGE is running. This allows the application server to optimize data access, e.g., by defining connection pools.

KNOWAGE supports both SQL, NoSQL and Big Data sources. The platform, at the time this document is written, supports the following dialects:

- Apache Cassandra[2]
- Apache Hive[3]
- Apache Spark SQL[4]
- HyperSQL[5]
- IBM DB2[6]
- Ingres[7]
- Microsoft SQL Server[8]
- MongoDB[9]
- MySQL[10]/MariaDB[11]
- Neo4j[12]
- Oracle 9i[13]/10g[14]
- Oracle Database Spatial[15]
- OrientDB[16]
- PostgreSQL[17]
- Teradata[18]
- Vertica[19]

In order to properly create a Data Source, the user has to insert additional attribute, in particular:

- **Label**: the identifier of the Data Source.
- **Description**: an optional parameter describing the Data Source.
- **Dialect**: the dialect used to access the database.
- **Read Only**: this attribute should have two values:
    - **Read Only**: the data source would not be used to write KNOWAGE temporary tables.
    - **Read and write**: the data source would be used to write KNOWAGE temporary tables.

[2] http://cassandra.apache.org/
[3] https://cwiki.apache.org/confluence/display/Hive/Home
[4] https://spark.apache.org/sql/
[5] http://hsqldb.org/
[6] https://www.ibm.com/analytics/us/en/db2/
[7] http://www.gdal.org/drv_ingres.html
[8] https://www.microsoft.com/en-us/sql-server/sql-server-2017
[9] https://docs.mongodb.com/
[10] https://www.mysql.com/
[11] https://mariadb.org/
[12] https://neo4j.com/
[13] http://www.oracle.com/pls/db92/homepage
[14] http://www.oracle.com/technetwork/database/database10g/documentation/
[15] http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html
[16] https://orientdb.com/
[17] https://www.postgresql.org/
[18] https://www.info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html#page/SQL_Reference/
[19] https://www.vertica.com/overview/

- **Use as cache:** the data source would be used to store temporary tables coming from other Read Only data sources. NOTE: only one Data Source should be configured as cache in a KNOWAGE installation.
- **Type:** Available options are JDBC and JNDI.
- **URL:** the specific URL of the data source.
- **User:** if authentication is needed, the user name.
- **Password:** if authentication is needed, the user password.
- **Driver:** the specific driver used to communicate with the Data Source.

Dataset Definition

The Dataset is the portion of data used to perform analysis. Datasets can be retrieved by querying the previously defined Data Sources or from external data providers. In particular, KNOWAGE supports, at the time this document is written, different dataset types:

- **Query:** performing a query over an already defined Data Source, following its dialect.
- **Flat:** this dataset type is also related to Data Sources. This dataset type will contain a complete table of a selected Data Source.
- **Query by Example (QbE):** this dataset type allows to query a database through a high-level representation of its entities and relations, called Business Model or Datamart. With this dataset type, the user can define dataset by querying a Datamart through an entirely graphical interface. The Datamart is built over an already defined Data Source by selecting which tables and/or relations would be part of the Datamart. For a detailed description of the Datamart definition, please refer to KNOWAGE CE's manual[20].
- **File:** by selecting this type, the dataset will contain the data coming from a CSV or an XLS file uploaded by the user.
- **Java Class:** the dataset is retrieved from a java class loaded into KNOWAGE's application server
- **Script:** the dataset is retrieved from a JavaScript or a Groovy[21] script.
- **Ckan:** the dataset is retrieved from a specific resource in a CKAN repository.
- **REST:** the dataset is retrieved from a RESTful API. This dataset specifically supports also NGSI APIs[22].
- **Solr:** the dataset is retrieved by querying a Solr[23] endpoint.
- **SPARQL:** the dataset is retrieved by querying a SPARQL endpoint

Dataset's fields' metadata could assume one of the following values:

- **Attribute**: the field can be used as a label or a grouping property in analytical documents. Analytical Document groups under a common concept the different types of documents that can be developed with KNOWAGE when performing an analysis, for additional details, please refer to KNOWAGE CE's manual.
- **Measure**: the field would be used as the value to be analysed into the documents.
- **Spatial Attribute**: the field would be mainly used as input for map widgets; this widget is currently in an early stage and it will be released in next versions of KNOWAGE CE[24].

Datasets should have a unique identifier, a name and a description. Moreover, the user can select the scope of the dataset and a category. The scope available options are *User*, *Enterprise* and

[20] https://download.forge.ow2.org/knowage/Knowage_6.x_CE_Manual.pdf
[21] http://groovy-lang.org/structure.html
[22] https://wirecloud.readthedocs.io/en/latest/development/ngsi_api/
[23] http://lucene.apache.org/solr/
[24] https://www.knowage-suite.com/site/ce-download/

Technical and this value is used to limit the visibility of the dataset depending of the specific role of other users. Category is a custom parameter and its available options depend on the categories defined by an administrator of the platform. Advanced Dataset configurations are, among others, the persistence of the dataset data into KNOWAGE's internal cache.

An advanced operation related to Dataset is the possibility to create a Federation of Datasets combining two or more heterogeneous datasets, coming from different data sources or external data providers, in a common model. The user would create the federation by creating the logical correspondence between dataset's fields (Figure 12) and the federated dataset will be used as a new model (Figure 13), which will allow to create new Datasets through the Query By Example front-end interface (Figure 14).

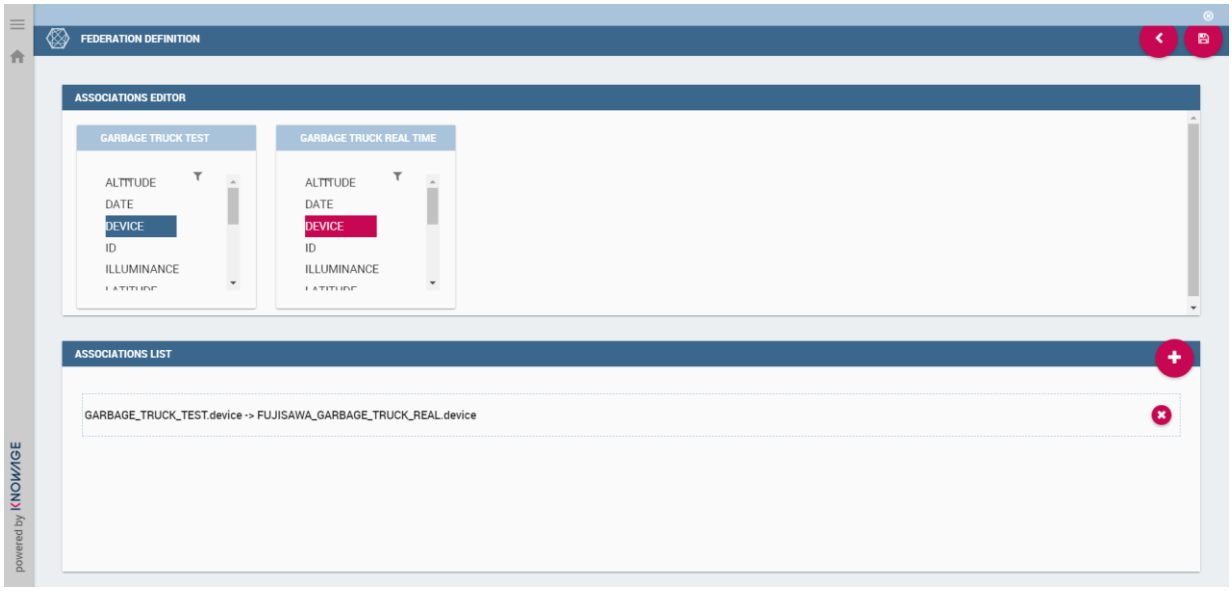

FIGURE 12: DATASET FEDERATION FIELDS SELECTION

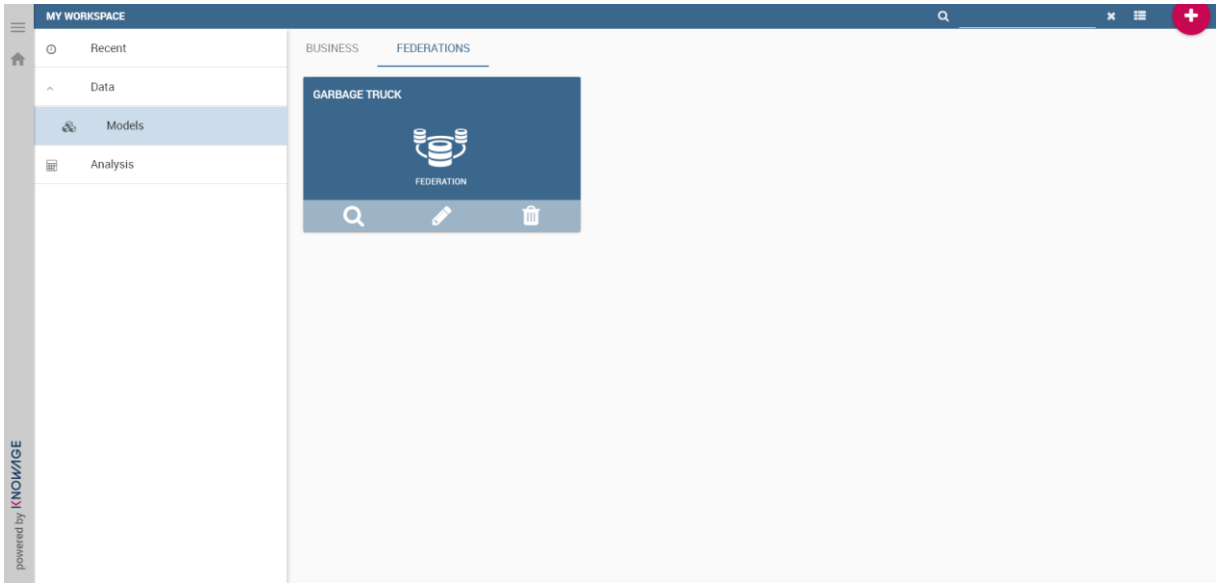

FIGURE 13: DATASET FEDERATION MODEL

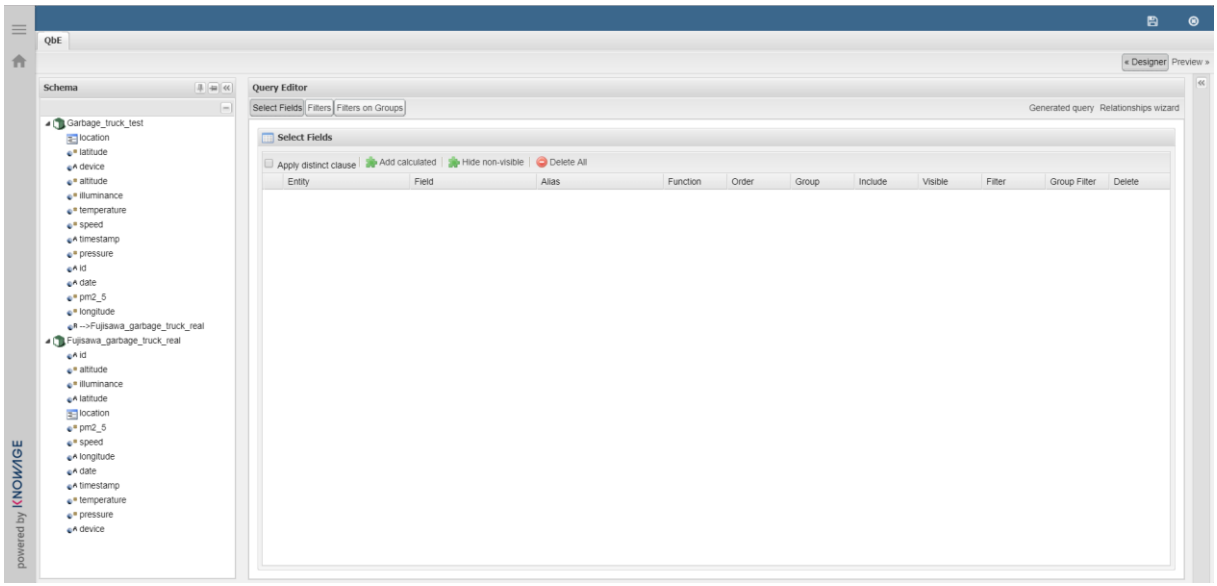

FIGURE 14: DATASET FEDERATION QUERY BY EXAMPLE

Filters to guide Analysis

Specific Dataset Types supports the definition of parameters. The Query dataset type is the main type where user can configure one or more parameters in order to build a parametric query to be executed every time a parameter changes its value. The available values of the parameter can be assigned through the definition of an Analytical Driver. The Analytical Drivers models a piece of data frequently used as a distinguishing criterion on the global data context. In particular, it models the values of a parameter, its scope according to end users' roles and its use mode and it is used during the execution of an analytical document. The main part of an Analytical Driver is a LOV. A LOV, or List Of Values, represents the possible values the Analytical Driver will use. In order to properly define an Analytical Driver, the user should follow these steps:

1. Define a parametric dataset (e.g. Query).
2. Create a LOV through KNOWAGE's GUI.
3. Create an Analytical Driver that uses the previously defined LOV.
4. Use the Analytical Driver during the execution of a document that uses the parametric Dataset.

4.2.2 Implementation details and Internal Architecture

KNOWAGE logical architecture (Figure 15) is layered on three main levels:

- **Delivery:** which manages the access to KNOWAGE's functionalities. The end user GUI or the RESTful APIs are parts of this layer.
- **Analytical:** which manages the analytical and behavioural functionalities provided by KNOWAGE.
- **Data:** which manages the access to data through the definition of Data Sources and Dataset coming also from external data providers such as CKAN repository or a REST interface.

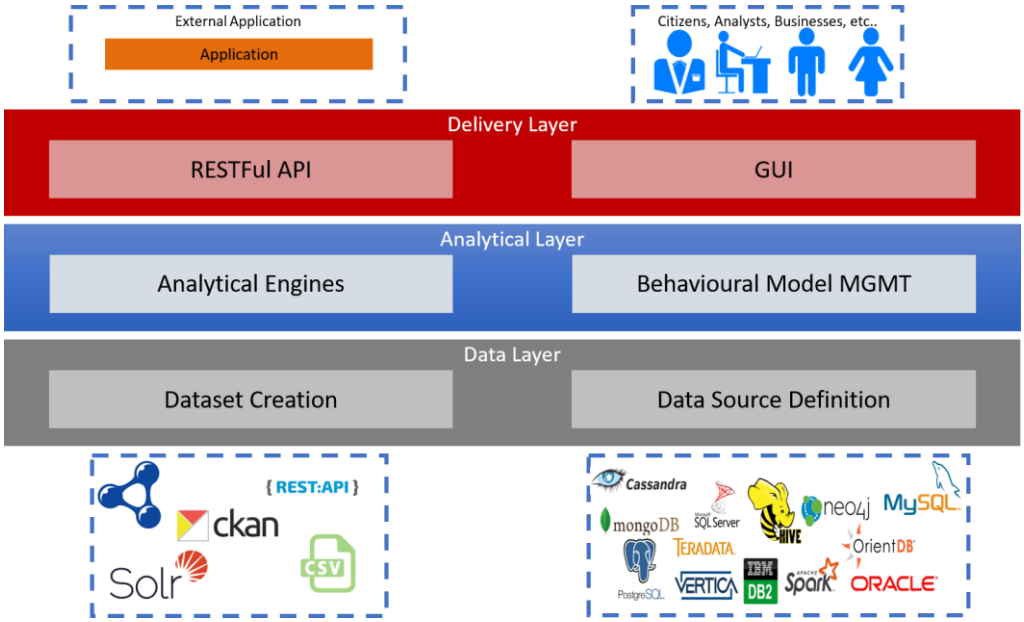

FIGURE 15: KNOWAGE LOGICAL ARCHITECTURE

The framework is developed with Java and its front-end application takes advantage of AngularJS[25] framework. Connection with Data Sources is managed with Hibernate[26] framework.

This section of the document covers the main functionalities provided by the Data layer. This layer of the architecture allows data access by connecting with Data Sources or by creating a Datasets storing data and metadata information.

4.2.3 Interfaces and Integration

KNOWAGE's functionalities can be accessed through its GUI. Depending on the role of the user and the authorization assigned by the administrator of the platform, the logged user would be able to manage or use different KNOWAGE's module and process. For instance, a simple user, with the default role, would not be able to create Data Sources or Datasets but he would only be able to execute documents.

KNOWAGE exposes also RESTful APIs in order to, for example, create, update or list dataset and documents. Enlightening on the Dataset resource, the main APIs provided by the tool are:

- **Returns the datasets**: which returns the list of the available datasets.
- **Add a dataset**: which is used to create the dataset.
- **Return the dataset**: which gives the detail of a single dataset.
- **Update the dataset**: which allows to update an existing dataset.
- **Delete the dataset**: which deletes the specified dataset.
- **Returns the dataset's content**: which accepts a list of query parameter and returns all of the dataset's field matching the query conditions.

The full list of the available APIs and their details are accessible in a dedicated Apiary[27].

The supported Data Source and Dataset types are described in Section 4.2.1.

[25] https://angularjs.org/
[26] http://hibernate.org/
[27] https://knowage.docs.apiary.io

4.2.4 Compliance tests

This section of the document covers the conformance and integration tests performed using KNOWAGE, BigClouT's CKAN repository and several SQL and NoSQL demo data sources.

BigClout's CKAN repository was used to test KNOWAGE's Dataset creation process. Different dataset types were used to map CKAN resources. Among the available dataset types, tests were carried out for: CKAN, REST and File dataset types. For the CKAN dataset the user should specify the URL of the resource he wants to use and additional attributes such as the file type (CSV or XLS). REST datasets were created taking advantage of CKAN datastore APIs. File dataset was created downloading a CSV resource from the repository and uploading the file into the platform.

SQL and NoSQL demo repository were used, on one hand, to test KNOWAGE's Data Source creation process. On the other hand, these repositories were used to test Query or Flat dataset types including parametric queries. The parametrical queries were subsequently linked to several Analytical Drivers in order to test this functionality during the execution of analytical documents. As for the SQL repositories, tests were performed using both MySQL and PostgreSQL instances. As for the NoSQL repository, a MongoDB database was used.

At the time this document is written, tests are performed in order to investigate about the integration between KNOWAGE and CDMI.

4.2.5 Candidate Use Cases to be supported

KNOWAGE, in the context of Big Data Analysis could support several use cases by giving the opportunity to access, aggregate and analyse data coming from heterogeneous data sources. In order to provide such functionalities, it is mandatory that KNOWAGE can access the data following the approaches described in Section 4.2.1, for instance creating a Data Source for the direct connection with a SQL, NoSQL or Big Data repository or a Dataset using an external data provider (REST API, CKAN, etc.). KNOWAGE could potentially support the following use cases:

- Grenoble's "*Use Case 1: Monitoring of Economic Impacts of Events*": analysing data and extracting statistics for the city economic development division and for the event organisers.
- Grenoble's *"Use Case 2: Monitoring of Industrial Estates"*: analysing data and extracting statistics for the city economic development division and for Innovallée Association.
- Bristol's *"Use Case 1: Smart Energy - predictive analysis of users' power consumption"*: comparing citizens' energy consumption and by providing the compatibility with NGSI APIs.
- Bristol's "*Use Case 2: Smart Mobility"*: merging information with other available sources (i.e. weather information). Moreover, KNOWAGE is compatible with NGSI APIs.
- Fujisawa's "*Use Case 1: Optimizing the incidence on local economy of Fujisawa"*: analysing data extracting useful information.
- Fujisawa's "*Use Case 2: Fine-grained city infrastructure management"*: analysing data extracting useful information.
- Tsukuba's *"Use Case 1: Provide tourism, traffic and environmental information in real time to visitors"*: analysing the trend of satisfaction rate, weather information and visitors' behaviour.

4.3 StreamOLAP

4.3.1 Description of Functionalities

StreamOLAP[28,29] is a data processing framework that allows OLAP (OnLine Analytical Processing) operation over data streams exploiting off-the-shell stream processing engine combined with OLAP engine. This system consists of two main processing components:

- **Stream processing component**: this component is responsible for continuously generating aggregation results at some selected aggregation levels using multiple queries. This component is in charge of providing stream processing features. Users can register queries in Jaql-like query language[30], thereby making it possible to continuously get filtered streams. Currently, it accepts JSON streams as input and provides output JSON streams, which can be reused by other systems to get more useful information.
- **OLAP component**: this component uses the in-memory results from the stream processing component to further compute the results at various aggregation levels that are not defined in the registered queries. Therefore, it can provide more insightful analysed information.

4.3.2 Implementation Details and Internal Architecture

Figure 16 illustrates streamOLAP architecture.

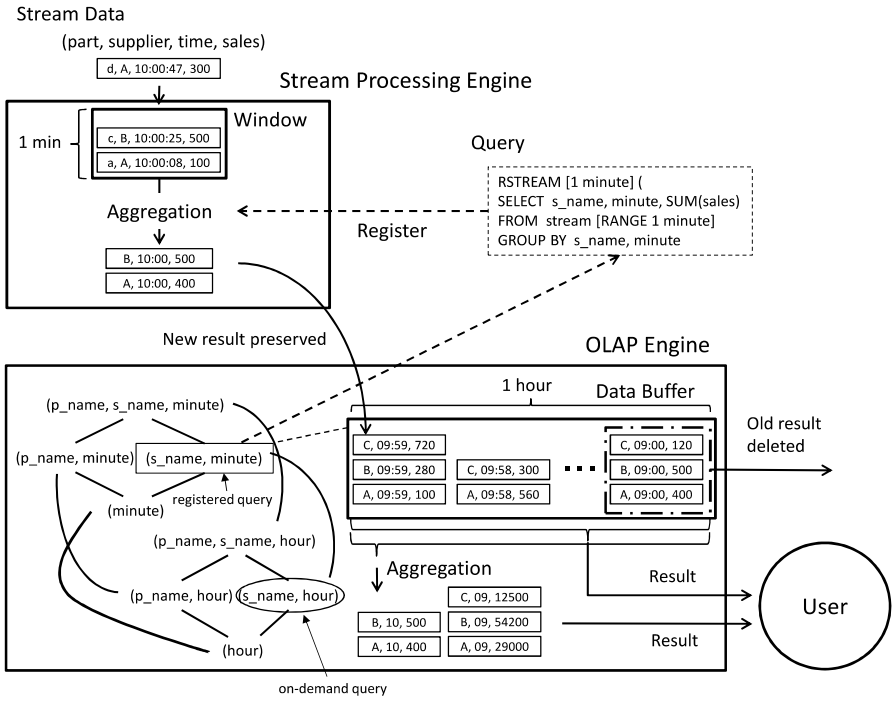

FIGURE 16: STREAMOLAP ARCHITECTURE

[28] K. Nakabasami, T. Amagasa, S. A. Shaikh, F. Gass and H. Kitagawa, "An architecture for stream OLAP exploiting SPE and OLAP engine," in Big Data (Big Data), 2015 IEEE International Conference on, Santa Clara, CA, USA, 29 Oct.-1 Nov. 2015.

[29] K. H. Shaikh S.A., "Approximate OLAP on Sustained Data Streams," Database Systems for Advanced Applications. DASFAA 2017, vol. 10178, pp. 102-118, 2017.

[30] K. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan and E. Shekita, "Jaql: A Scripting Language for Large Scale Semi-Structured Data Analysis," PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, vol. 4, no. 12, pp. 1272-1283, International conference on very large data bases; VLDB 2011.

As discussed above, it has two processing engines: 1) a stream processing engine and 2) an OLAP engine. The stream processing engine is responsible for continuously generating aggregation results at some selected aggregation levels using multiple CQL queries (Called Registered Queries) over streams whenever a new tuple arrives. The results are stored in the in-memory buffer, called data buffer, in the OLAP engine for immediate or future references from users. Meanwhile, as for those aggregation levels that are not assigned any CQL query, the OLAP engine computes the result using available results by (non-streaming) relational aggregation queries (called on-demand queries). The results are returned to users.

4.3.3 Interfaces and Integration

4.3.4 Interface

A Web user interface of this system is provided in order to allow users to submit the queries and extract results at various aggregation levels (e.g.: roll up, drill down, sum, min, max, etc.). The results are in form of tables, charts, and graphs. In the context of BigClouT and smart city management, it can provide insightful analysis of city data so that administrators of a smart city can monitor current city status more efficiently. StreamOLAP interface is shown in Figure 17.

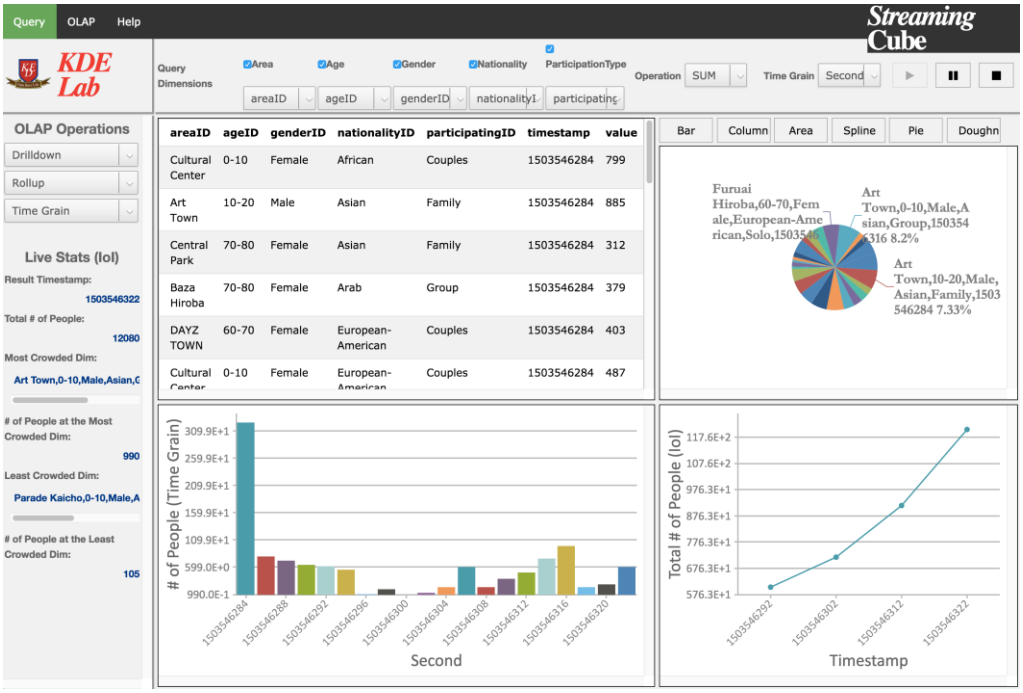

FIGURE 17 STREAMOLAP INTERFACE

4.3.5 Integration

I/O manager of the Data Event Processing component is responsible for dealing with and accepting data streams from the City Resource Access module (Figure 4). City data from the City Resource Access module is accessible by either HTTP request or REST API. Thanks to the well-defined I/O manager, Data Event Processing can connect with City Resource Access module with minimum customisation to the I/O manager by either:

1. Direct Connector: Directly add functions to the I/O manager of the Data Event Processing to read/access city data from the City Resource Access module.

2. Socket Stream Connector: The City Resource Access module directly sends the city data to the Data Event Processing component.

Both approaches can be done either in a synchronous way (client/server mode) or in an asynchronous way (publish/subscribe mode) by requesting or subscribing for city data from the City Resource Access module.

Notice that, the above methods require the registration of schema of the city data to the I/O manager of Data Event Processing. The schema is in JSON format textual document. In addition, I/O manager is also responsible for dispatching the output of Data Event Processing to other logical subcomponents of the City Data Processing.

4.3.6 Compliance tests

StreamOLAP can efficiently process large-scale city data. Stress-tests were performed, and the results were shown in Figure 11.

4.3.7 Candidate Use Cases to be supported

The candidate use cases to be supported by this functionality are the same with the ones presented in Table 1.

4.4 DeepOnEdge

How can we inspect city conditions at low costs? City infrastructures, such as roads, are elements of great importance in urban lives. Roads require constant inspection and repair due to deterioration, but it is expensive to do so with manual labour. Therefore, these works should be done automatically so that the cost of inspecting or repairing becomes cheap. While there are several works to address these road issues, our study focuses on official city vehicles, especially garbage trucks, to detect damaged lane markings (lines) which is the simplest case of road deterioration. Since our proposed system is implemented on an edge computer, it is easy to attach our system to vehicles. In addition, our system utilizes a camera, and since garbage trucks almost run through the entire area of a city every day, we can constantly obtain road images covering wide areas. Our model, which we call Deep on Edge (DoE), is a deep convolutional neural network which detects damaged lines from images. In our experiments, to evaluate our system, we first compared the accuracy of line damage detection of DoE with other baseline methods. Our results show that DoE outperforms previous approaches. Then, we investigate whether our system can detect the line damage on a running car. With this demonstration, we show that our system would be useful in practice.

4.4.1 Introduction

The road is one of the most important infrastructures of a city in planning and development. For instance, people usually use them for going somewhere or for planning land utilization to enrich their livelihoods. However, many roads need repairing since most of them are built in periods of rapid economic growth and have been deteriorating since. Thus, to inspect their condition for road repair, the city administration needs to employ people for constant inspection. Yet manual road inspection is expensive and takes a lot of time; for instance, in order to detect the damage or blur of road markings, people have to check by eye, whose ability has certain limits. In addition, in certain regions such as Japan, public funds for road inspection have been reduced due to current

societal conditions. In short, manual road inspection and repair is not enough for sufficient maintenance.

Most previous work has therefore focused on making the cost of road inspection cheaper to increase sustainability. In contrast, we aim to detect the damage or blur of white lines. Detecting the damage of white lines is difficult to do using smartphone accelerometers. Thus, we use a camera to take pictures. If we use participatory sensing as well as and collect the images, however, the cost issue still remains due to the cost of platform introduction and labour.

To tackle this issue, we focus on city vehicles, especially garbage trucks. Since garbage trucks run their services every day and cover a whole area of the city, if the garbage truck equips a camera and takes pictures of roads, we can obtain road images from the whole area. Furthermore, we do not have to pay additional costs for labour or facilities. However, the number of running garbage trucks is so large (e.g., hundreds of trucks) that it is troublesome to storage and manage image data in a centralized way. Simultaneously, if we upload an image every time a camera takes pictures, it would take great communication costs and bandwidth. In summary, our goal is proposing a system that can be attached to garbage trucks and detect white line damage on the spot.

In order to achieve our goal, we introduce Deep on Edge (DoE), which integrates edge computing and deep neural networks. The overview of our system is depicted in Figure 18. DoE consists of an edge computer (e.g., Raspberry Pi 3) with a camera to be attached to garbage trucks. When DoE detects line damage, the results are reported and sent to cloud computing. Then, we use those reports and understand city condition. We treat the task of line damage detection as a classification problem. We train a convolutional neural network (CNN), a type of deep neural network, on labelled images on a GPU server. At inference time, DoE is loaded on an edge computer and outputs a discrete probability distribution, assigning each image a likelihood that the white line in the image is damaged. There are some constraints to use DoE on edge computers due to restricted performance, while on the other hand we do not have to consider the number of parameters or inference speed when we use DoE on a high-performance computer. So to use DoE on edge computers, we design the CNN architecture to be as small as possible but to keep the accuracy high. To evaluate DoE, we compared it with baselines on the line damage detection task. As a result, DoE outperforms baselines on this task while reducing the number of parameters. Simultaneously, we visualise DoE activation so that we can understand how it has learned to detect line damage.

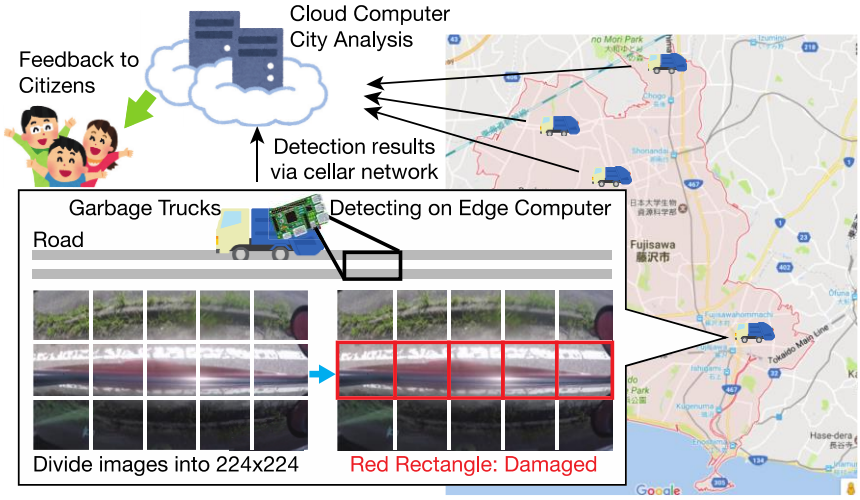

FIGURE 18: SYSTEM OVERVIEW. EACH CITY VEHICLE RUNNING IN THE CITY DETECTS WHITE LINE DAMAGE. THE CLOUD COMPUTER AGGREGATES THE RESULTS FROM THEM AND MONITORS THE CITY.

4.4.2 *Summary of our lane marking dataset*

For detecting road damage, we focus on the damage or blur of white lines, which we assume is the most common type of lane marking. To collect line images, we attached a normal camera, which can film by 60 fps, on a side of a passenger car so that the camera always films the line. Then, we drove the car within 50km/h for four days from March 30th, 2016 to April 2nd, 2016 in daytime. Note that it was sunny days. While we got videos in which each frame is 1024 * 768 pixels after filming, we randomly cropped frames into 224 * 224 pixels. This cropping was done for reducing the training time until the model convergence and allowing the model focus on the line damage. One participant annotated those cropped images with three kind of labels; damaged line, undamaged line, and no line. After the pre-processing described above, we obtained 43000 images of lines. At our experiments, we divide the dataset to 35000:8902. The examples of our dataset are shown in Figure 19 and described in detail in Table 2.

FIGURE 19: DATASET SAMPLES

TABLE 2: THE DATASET WHICH WE COLLECTED, PREPROCESSED AND ANNOTATED.

Class	Type	Undamaged	Damaged	No line	Total
Binary	Train	10696	14304	–	25000
	Test	3829	5073	–	8902
Trinary	Train	15445	11568	7987	35000
	Test	3932	2957	2013	8902

4.4.3 *Deep on Edge System*

We pose the task of line damage inspection as a classification problem. For this, we use a dataset of images of lines with three kinds of labels described in the previous section. The input to DoE are image pixels and the target output is a one-hot vector encoding those labels. Given an image, the output of this model is a probability distribution describing the extent of road damage. The advantage of outputting a probability distribution is that this gives the model the ability to give specific scores to a line image, taking out the necessity of a human expert to give specific ratings.

Our Model

In order to detect line damage from images, we adopt a convolutional neural network, which is a special type of feedforward neural network or multi-layer perceptron[31] and works well with two-dimensional images. We design our CNN by referring to the VGG16 architecture[32]. VGG16 is one of the major CNN architectures which was used to win the ImageNet Large Scale Visual Recognition (ILSVR) competition in 2014, although it has been outperformed by great advances such as Inception[33] and ResNet[34]. VGG16 only uses convolutional layers with 3 * 3 kernels and pooling layers with 2 * 2 kernels. This feature is very significant for DoE since the size of the model is required to become as small as possible to work on edge computers. Given an input image X of width w, height h and c colour channels (usually RGB channels) represented as $X \in R^{w \times h \times c}$ at each convolutional layer, it is convolved with d sets of local kernels $W \in R^{w \times h \times c \times d}$ and bias $b \in R^{d}$ is added:

$$h = \varphi(W * X + b), \quad (1)$$

where * denotes a convolution operation and φ is a non-linear function that we use the rectified linear unit (ReLU, max{0, x}). Max-pooling, a form of non-linear down-sampling, is applied to the output of the convolution. Max-pooling partitions the input into a set of non-overlapping rectangles by the kernels and outputs the maximum value in each sub-region respectively. This operation is very useful because it reduces the dimensionality of a high-dimensional (high-resolution) output of the convolutional layer and summarizes the activations of neighbourhood features so that model becomes robust to local perturbations. Since our input images are filmed from a driven car, the location of lines in the image are not fixed. DoE is built by several alternating convolution layers and max-pooling layers.

In VGG16, the output after some convolution layers and max-pooling layers is flattened for the input of to the following layers, which are fully-connected. If the shape of the output of convolution is $R^{w \times h \times c}$ and the output dimension of the next fully-connected layer is d, the number of parameters in that FC layer becomes $w*h*c*d$. This is a problem when the size of the input image is large, since the larger the image size is, the larger the number of parameters becomes. To avoid the increase in number of parameters, we use global average pooling[35] instead of flattening. Applying global average pooling allows the number of parameters in the FC layer c * d to be independent of the input image size.

At the last layer of DoE, the output is a probability distribution over the possible conditions of the road. The model of the DoE architecture which we used in our experiment is depicted in Figure 20.

[31] Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.
[32] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.
[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," arXiv preprint arXiv:1512.00567, 2015.
[34] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385,2015.
[35] M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv preprint arXiv:1312.4400, 2013.

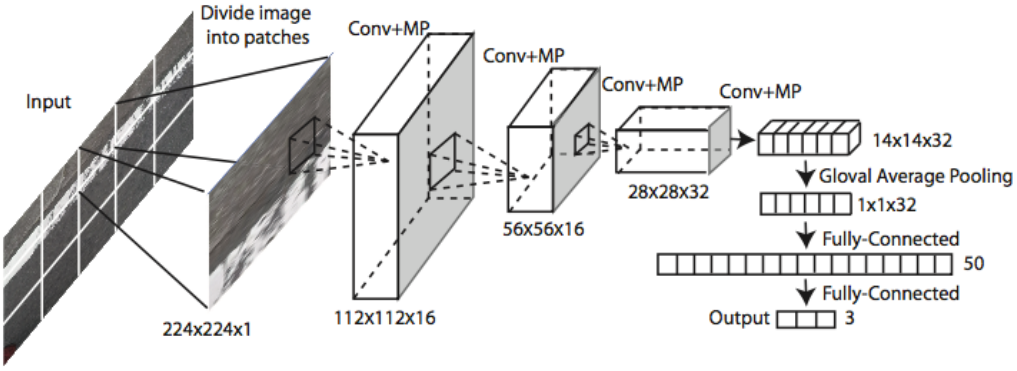

FIGURE 20: OUR MODEL OF DOE ARCHITECTURE

Implementation for Practical Use

While DoE is trained with the road images of size 224 *224, the size of images from a video camera is much bigger than that. Although our DoE model can take any image resolution, our preliminary experiment showed that DoE cannot detect the line damage accurately at any resolution. In order to tackle this issue and use DoE for practical use, we implement a module that divides the input image into 224 *224 sub-regions and reshapes these sub-regions to $X \in R^{n \times 224 \times 224}$ where n is the number of sub-regions. Even if n is very large, DoE is able to process it all at once. For instance, if the size of an input image is 1280*720, the number of sub-regions becomes (1280/224)*(720/224) ≈ 15 and the input to model $X \in R^{15 \times 224 \times 224}$. Although our module crops out the top, bottom, and right sides of the image, this is not a problem because of two reasons: (a) the top and bottom sides of the image usually does not contain the road and (b) the road contained on right side is contained in the next input image. Figure 20 also shows this module.

4.4.4 Experiment

In this section, we show two kinds of experiments. First, we compare DoE with baselines which are used in previous works to evaluate DoE. Then, we examine whether DoE is fit for practical use.

Accuracy Comparison

In order to evaluate DoE and its architecture, we compare it with previous work. Although Maeda et.al[36] classify the degree of road condition into three types: *smooth* (no-damaged), *need repair* and *not need repair*, its actual classifications are difficult to distinguish, as different outputs are produced from visually similar images. To make this problem more interpretable, we simplify this task as binary classification problem: the road which is photographed in given images is whether damaged or not. Therefore, we used the dataset we use consists of images labelled *damaged* and *undamaged* in Table 2. As baselines, we adopt two kinds of methods. The first method tests classic machine learning algorithms: a Support Vector Machine classifier (SVM) that can achieve good performance at binary classification, and a random forest which can detect the line damage[37] as

[36] H. Maeda, Y. Sekimoto, and T. Seto, "Lightweight road manager: smartphone-based automatic determination of road damage status by deep neural network," in Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems. ACM, 2016, pp. 37–45.

[37] T. Kawasaki, M. Kawano, T. Iwamoto, M. Matsumoto, T. Yonezawa, J. Nakazawa, and H. Tokuda, "Damage detector: The damage automatic detection of compartment lines using a public vehicle and a camera," EAI MOBIQUITOUS2016IWWSS2016, 11 2016

well as our work. The second method is a deep neural network. We choose the AlexNet[38] which won the ILSVR competition in 2012, and has been used quite active since[31]. Further, since the aim of our study is road detection on an edge computer, the smaller model is desirable and we also examine alternative models: AlexNet--(d) and AlexNet--(e).

Before training DoE, we initialize the weights of DoE with random values and use the Adam stochastic gradient descent algorithm[39] with a learning rate of 0.0005, a momentum of 0.9 and a batch size of 32. Meanwhile, those of baseline networks use respective values of 0.0001, 0.9 and 100. We then trained models with early stopping, which is a training procedure that stops training if the error on the validation set stops decreasing.

TABLE 3: ACURRACY COMPARISON ON THE LINE DAMAGE BINARY CLASSIFICATION TASK. THE NUMBER OF PARAMETERS IS THE SUM OF WEIGHTS AND BIAS.

Method	Acc.	AUC	Recall	Pre.	F1	Params.
Linear SVM	82.4	0.82	0.87	0.83	0.85	–
Random Forest [7]	84.0	0.83	0.91	0.83	0.87	–
AlexNet [14]	92.5	0.9833	0.92	0.92	0.92	58000K
AlexNet–(d) [6]	92.5	0.9845	0.92	0.92	0.92	1680K
AlexNet–(e) [6]	92.7	0.9859	0.93	0.93	0.93	913K
DoE (ours)	**94.1**	**0.9894**	**0.94**	**0.94**	**0.94**	**18K**

Table 3 shows the experiment result. DoE outperformed baseline methods, even though the number of parameters is quite less than others. This result shows that deep architectures do not necessarily have good performance in computer vision tasks, even if it has been reported as good architecture. In short, it is necessary to tweak model architectures for specific tasks.

TABLE 4: CONFUSION MATRIX OF DOE.

Number of Parameters 18171		Prediction			Recall
		Undamaged	Damaged	No line	
Ground Truth	Undamaged	2795	162	0	0.945
	Damaged	196	3734	2	0.950
	No line	0	1	2012	1.00
Precision		0.945	0.950	0.999	0.980

Practice Investigation

For practical use, we examine whether DoE is able to detect line damage from an actual image from a camera. For this, we retrain DoE from a binary classification problem to a 3-class classification problem; *undamaged* (no-damaged), *damaged*, *no line*. When we use DoE that solves

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.
[39] https://arxiv.org/pdf/1412.6980.pdf

a binary classification problem, it may cause false detection when there is no line. The result confusion matrix of 3-class classification is shown in Table 4 and actual detection in Figure 21. As a result, DoE can classify the road condition with 98% accuracy. Furthermore, in the case of Figure 21 (a) (b), DoE classifies the patches perfectly. Note that in (b), at the location of the yellow font, the left upper patch is classified as ``undamaged'' correctly, whereas patches right side hand of it is classified as *damaged*. On the other hand, DoE misclassifies *undamaged* patches as *damaged*. This might happen if the line is dirty or something is on the line (e.g. the shoe is photographed in Figure 21(d)).

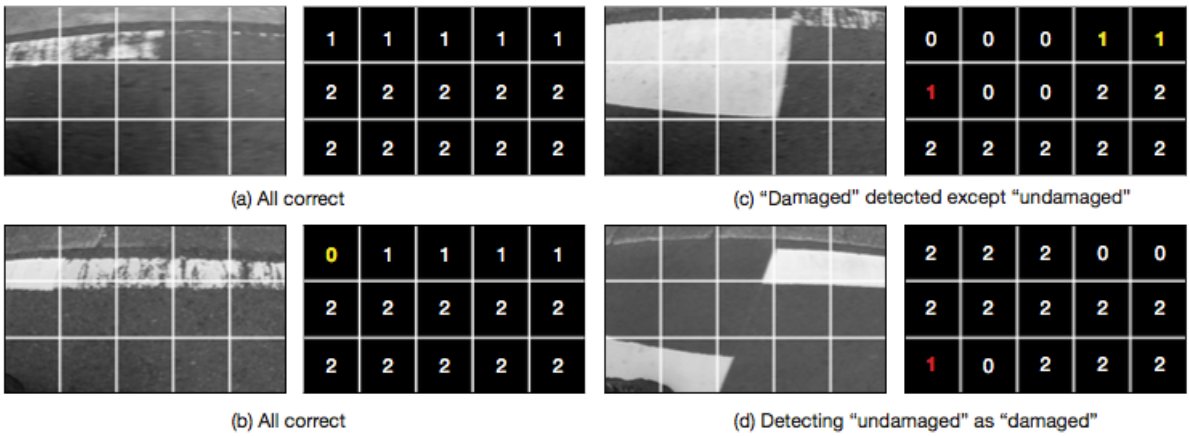

FIGURE 21: THE RESULT OF THE LINE DAMAGE DETECTION WITH ACTUAL IMAGES. EACH NUMBER DENOTES THE CLASSES, RESPECTIVELY; 0: UNDAMAGED, 1: DAMAGED, 2: NO LINE. (A) (B) DOE CLASSIFIES ALL PATCHES CORRECTLY. (C) ALTHOUGH DOE MISTAKES CLASSIFYING ``UNDAMAGED'' AS ``NO LINE'' (AT RED FONTS), IT CORRECTLY DETECTS DAMAGE AT YELLOW FONTS.

5 Machine Learning, Predictive Modelling and Decision making

5.1 General Description

The presented service is based on a distributed architecture which effectively handles sensor data and Open Data and uses state of the art technologies and tools to deliver recommendations applied in applications in the convergence of Internet of Things, Big Data and Smart Cities. It executes queries in the Graph Database to receive answers to simple or complex questions and produces a recommendation based on stored data and dynamic information.

5.2 Recommendation Service

5.2.1 Description of Functionalities

This service provides recommendations to end users/citizens in several application settings such as transportation management, energy consumption, public safety, supply chain management, tourism services, air and sound pollution management and many others.

5.2.2 Implementation details and Internal Architecture

As reported in section 5.2.1, *Recommendation Service* can be applied in different domains. To better explain its implementation and its internal architecture, a scenario about a *smart-heating* is taken into count.

As shown in Figure 22, the Recommendation Service consists of three independent components:

i) the Recommender Application with a Front-End and Back-End which handles the interactions with the user and delivers a heating schedule which is automatically applied to the flat;

ii) the Neo4j Graph Database;

iii) the IoT Node-Red Flows component which wires together the different hardware devices, APIs and web services, connecting the distributed components, sensors into a common IoT application. The latter component mainly contains the two flows presented below.

It is important to underline that all three components are independent; this represents an important feature for possible extensions and scalability.

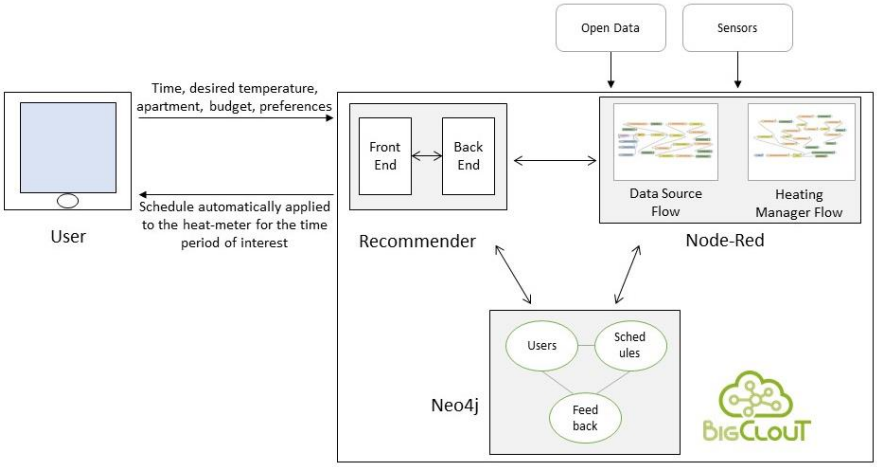

FIGURE 22 OVERVIEW OF THE PROPOSED ARCHITECTURE

Data Source Flow

The first Node-Red flow, called the "Data Source Flow", handles the input of the data from different sources. In order to effectively transmit data that are being exchanged between the sensors as well as the main system, the MQTT protocol[40] is utilised. MQTT is a lightweight publish and subscribe messaging protocol for use on top of the TCP/IP protocol, and is ideal for use with low power sensors with limited resources. MQTT is based on the principle of publishing messages and subscribing to topics. More specifically, in our scenario, sensors behave as clients who connect to a broker and publish messages to topics, while the broker enables the connection, acting as a common interface. The data in the MQTT broker are transmitted using a simple JSON format. An example of a JSON message used in a smart-heating scenario, its format and included information is presented below:

```
{
  "estate":"Dalehead",
  "hid":"cbe3WqEPlM",
  "ts":1483729805,
  "heatmeter":{
    "returnTemp":"20.5",
    "flowRate":"297",
    "flowTemp":"75.7",
    "ts":"1483729805",
    "instant":"9280",
    "cumulative":"30503"
  }
}
```

Two kinds of devices are being used in this implementation, sensors and heat-meters. The information includes the unique identity of the house which transmits the data and information about the type of the device, which can be either capturing information about the external temperature or humidity. The message also includes the actual numerical values the device captures. We have implemented a Node-Red MQTT node to effectively gather, process and republish data to the connected services or brokers when required. Before actually forwarding them, the Data Source Flow performs a first level processing of the data in order to calculate specific failsafes for each of the houses.

More specifically, a Generic Data failsafe is being calculated, counting the number of measurements we receive from each house in a specific timeframe, ensuring that they are above a certain numerical threshold, thus ensuring the validity of their frequency. In addition, a series of failsafes concerning the temperature data are being calculated, such as failsafes to locate measurements below the absolute zero or above a selected high temperature (70 ^{o}C), which would

[40] http://mqtt.org/

indicate a malfunction in the sensor. These failsafes are being used to exclude the specific house from the processing, resulting in a warning statement and ensuring the validity of the data. The flow generates similar failsafes for the heatmeter data.

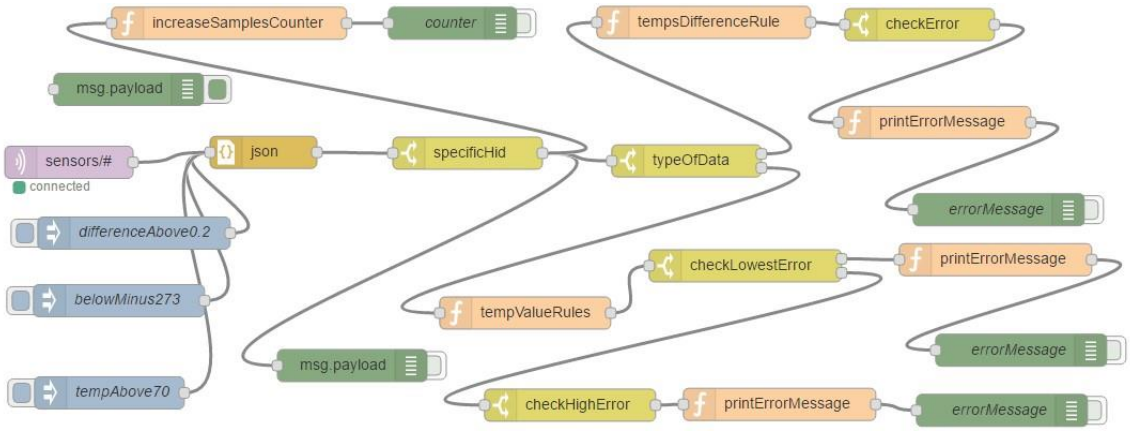

FIGURE 23: DATA SOURCE NODE-RED FLOW HANDLING INPUT FROM SENSORS AND OPEN DATA, COMPUTING AND MONITORING SPECIFIC FAILSAFES AND ALERTING THE USER ACCORDINGLY

Heating Schedules Management Flow

Following the initial data processing, the information is being forwarded to the Heating Schedules Management Flow, whose goal is the control of the heating schedule of each house's heat-meter. To achieve this, the system takes into consideration the provided users preferences and input. HTTP Requests have been implemented with the corresponding Node-Red nodes enabling user interaction with our system as well as with front-end applications (i.e. Mobile Applications, Web Applications, or any other type that can issue HTTP requests).

This enables an efficient storing in the flow of parameters related to user preferences such as:

i. the Illness Indicator, which indicates whether the user is ill;
ii. the Away Indicator, in case the user is out of house;
iii. the Failure indicator, in case there is a system bug;
iv. the desired temperature with the Auto Calculate Temperature indicator, in case the user prefers the system to calculate the ideal temperature for a specific timeframe.

In addition, the Heating Schedules Management Flow controls the houses' heat-meter schedule, by allowing the user to add a new heating schedule for a specific timeframe, by selecting a desired date, starting time and ending time. The flow also provides multiple schedules support, with handling of potential overlapping. Subsequently the flow continuously monitors the schedule information in order to handle the execution when the next desired starting time arrives. The flow also takes into consideration the user preferences as well as the failsafes calculated in the Data Source Flow, thus achieving flow integration, and produces the proper error messages, or adjusts the heating schedule accordingly.

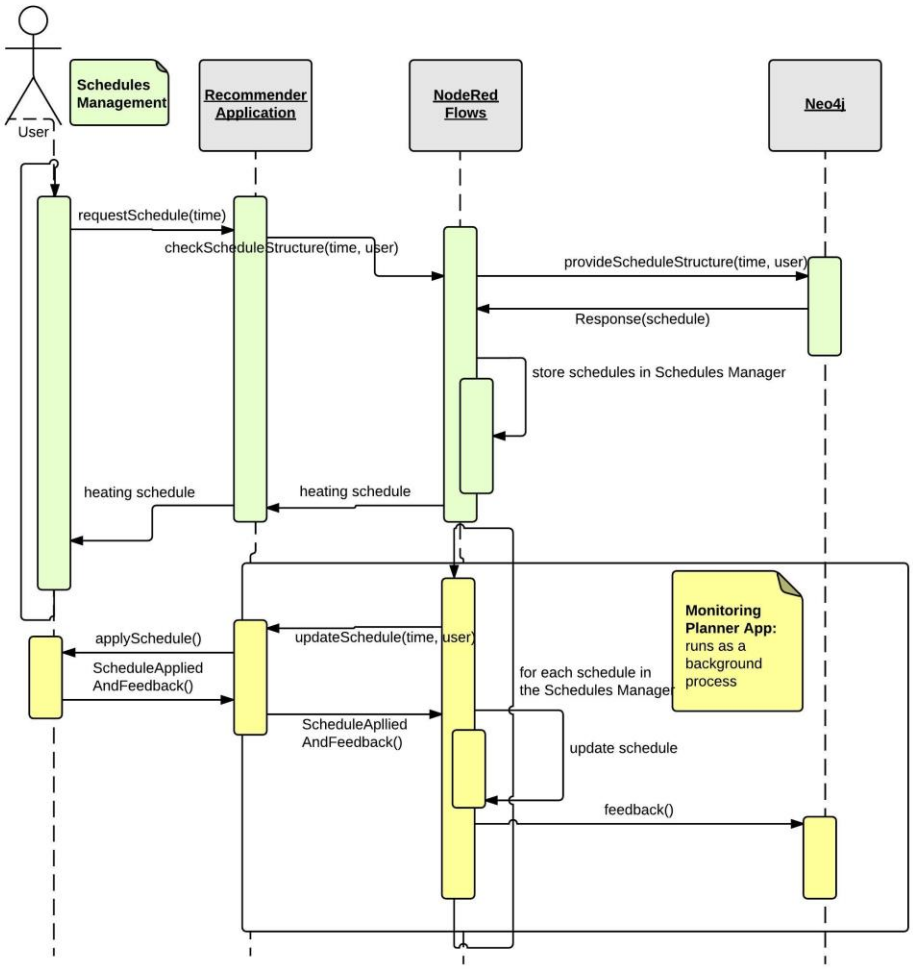

FIGURE 24: HEATING SCHEDULE MANAGER SEQUENCE DIAGRAM

Graph database Modelling - Heating schedule management

The implementation is based on a state-of-the-art smart city application, which is based on open data captured by the city IoT infrastructure and user generated content in terms of apartment characteristics and resident's consumer profile. Such an application is regarded as a smart city application for a particular city context.

The open data that are being deployed refer to:

i) weather conditions as collected by the weather stations (measuring wind speed, rainfall, humidity and temperature);
ii) weather forecast for a certain period;
iii) apartment specifications (surface area, orientation, borough);
iv) user preferences in terms of consumption (default, economic, illness) along with a dynamic rating mechanism where users give current feedback giving a score (from 0 to10) depending on whether they find the current schedule appropriate for the specific apartment and outdoor weather conditions.

The user provides details regarding the flat, the available budget, the scheduling period, etc. Another optional input is the desired temperature for the specific period. The system, taking into consideration these features, makes complex queries to a graph database to collect related information and produces personalized recommendations for the specific user.

One of the most remarkable features of a graph database, which is also crucial for our implementation, is its effectiveness into handling big volumes of information with real-time response to queries. Our system, based on a highly scalable graph database, processes in real-time simple and complex questions such as "What is the temperature on a specific day/time?", "What is the level of humidity and rainfall on a specific timestamp?" by rapidly traversing the graph while reading and processing values in nodes, relations and properties. Similarly, it handles other questions related to the historical data of users regarding past schedules and environmental conditions. Questions about ratings on past schedules the user has applied are effectively handled, by traversing the graph as shown in Figure 25, where the ratings of users are depicted.

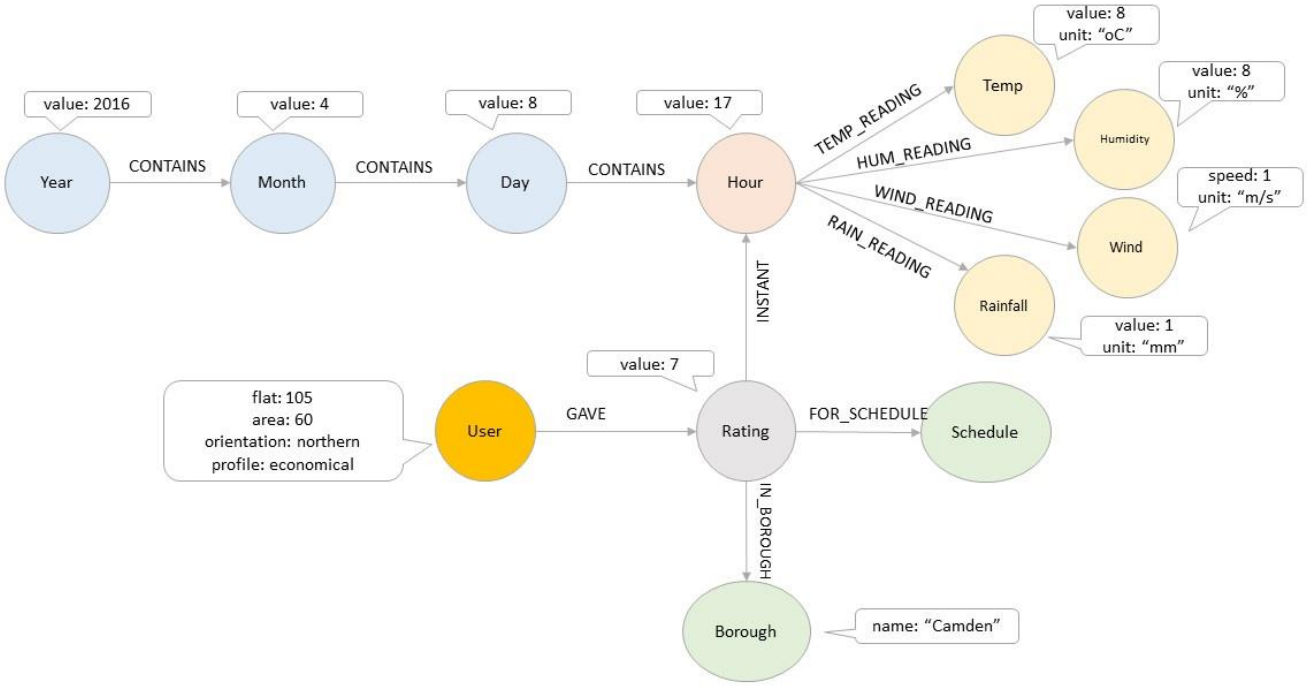

FIGURE 25: GRAPH DATABASE MODELING USING TIME TREES AND HANDLING BIG VOLUMES OF OPEN DATA COMING FROM THE CITY

Graph database Modelling - Overall modelling approach

A significant advantage of graph models is that they depict entities and relations in the same way as we think of them. Moreover, these models are forming a view on the queries we would like to implement in the graph database. The modelling process and approach in graph databases can be regarded as equivalent to the approach of creating graph structures that reflect the queries we would like to answer. "Users" or "Ratings" for example comprise the nodes of the graph, "names" are the attributes of the nodes, verbs such as "likes" depict the relations that link the nodes, and whatever refers to such verbs is regarded as the attributes of the relations. An interesting way to model time in Neo4j is through the use of time trees. In this approach, nodes represent years, months, and days, etc. while every node contains an attribute "value".

By forming the time trees, we can link particular events or other measured data on these trees. In our example, we link measurements of weather conditions (comprising of temperature, wind speed, rainfall and humidity) which have been collected in the Camden borough of London and are provided as open data[41]. In addition, real data originating from houses in the same borough have been utilised. With the consent of the owners, these houses were equipped with special humidity and temperature sensors, as well as special heat-meters able to transmit the users'

[41] http://www.londonair.org.uk/LondonAir/Default.aspx

desired temperature The data have been fed into our graph database and linked together in an efficient and constructive way allowing the user to ask a multitude of questions such as: "what is the temperature", "what are the weather conditions", etc. At the same time users have been added in the database along with their flat's characteristics, previously applied heating schedules and the corresponding ratings.

We begin the implementation of our system using Neo4j's query language Cypher, a declarative, SQL-inspired language for describing patterns in graphs visually. It allows us to state what we want to select matching a specific pattern, insert, update or delete data from our graph without requiring us to describe exactly how to do it[42,43]. Firstly, we create the time trees, which constitute the keystone of our model. In order to achieve faster data retrieval and improved performance, we use indexes, a feature provided by Cypher, created once over a property for all nodes that have a label. Cypher automatically manages the update by the database whenever the graph is changed. Then, we populate our model by importing the temperature and weather data into the graph database linked to the suitable time nodes, adding additional integration layers when required e.g. an additional layer is required for wind level and speed. We should note that Neo4j offers great data visualisation options allowing us to view all the stored nodes, relations and information, giving real-time insights into how data nodes are related, facilitating the development and evaluation process.

The system provides a user-friendly way for users' registration in order to deliver heating recommendation services. When a user is registered to the system, a new unique node is created with the apartment characteristics stored as properties. Afterwards, the user inserts preferences and historical data, and the corresponding nodes and relations are created in the graph database. Our model includes a weight scale for all heating schedules from 0 to 10 instead of a simple binary field. This way the user gives a more detailed feedback and in the end a more personalised recommendation is delivered. The next step is the addition of evaluations regarding past and real-time schedules. Every evaluation node is connected to the user who provided it, the corresponding node in the time tree and borough. This way a specific rating is connected to all the related information of our model.

Recommendations

Our proposed recommendation service exploits similarity metrics among vector representations, users' preferences, previous schedules applied and ratings to deliver in real-time recommendations in the form of heating schedules. The user receives the top scheduling recommendations and chooses one to be automatically applied for the following hours of interest. For example, a resident of a 60 m^2 flat in the Camden borough with northern orientation wishes to receive an economical heating for the following ten hours. Our service provides the most suitable schedules after tracking the closest users, historical data, the number these schedules were applied and past evaluations. The user chooses the best one and afterwards optionally provides feedback in the form of a rating or suggestion for alteration if required.

Recommendation systems produce suggestions through various approaches, while they often take into consideration the dynamic context to establish relations among users and target objects. The production of effective recommendations is based on the comprehension of these relations and their quality as well. Graphs are perhaps the most suitable structure to represent dense connected data structures. By storing and studying these data using graph databases, an application is allowed to exploit and demonstrate in real-time the impact of users' actions and not

[42] https://neo4j.com/developer/cypher-query-language/
[43] I. Robinson, J. Webber, and E. Eifrem. Graph databases: new opportunities for connected data. " O'Reilly Media, Inc.", 2015.

just take advantage of predefined results of pre-existing data. A widely applied technique in collabourative filtering recommendation systems is identifying the closeness among different users with similarity metrics. To this direction, we model application users as vectors and then compute the distance between them. This way we locate users who are closer to the target-user, study their behaviour and make the corresponding recommendations.

A user is represented as a vector, considering the flat's characteristics, the consumer's profile and preferences: User = <Surface area, Borough, Orientation, Consumer profile, Preferences> where consumer profile and preferences are related to budget, current state (i.e. illness, economical) and desired temperature, if it is provided. For example, the previously mentioned user could be represented as a vector <60m2, Camden, Northern>, whereas the additional information such as consumer profile, starting and ending time, desired temperature are also taken into consideration to find the closest users. By executing queries in the graph database, existing solutions for the closest users are returned with the evaluation and the times they were applied. The recommendation service exploits these data and delivers the top schedules for the particular period. Our approach is based on vector representations and we transform the problem of finding similar users into the equivalent problem of finding the closest vectors representing users. To this direction, we compute distances between vectors using metrics. The first metric is cosine similarity between two vectors and takes values in the range [-1,1]. The second presented metric is Euclidean distance and we compute the distance between two points in the n-dimensional space.

We form complex Cypher queries delivering recommendations in real-time using the Euclidean distance (or Cosine similarity) to compute the similarity among a target-user and all users in the graph database who have ranked activities and go through the following steps:

1) find the time node and the weather conditions for this particular moment;
2) aggregation of all users who have scored a schedule in the time window under study;
3) find all existing schedules for similar weather conditions in the past;
4) find the users, the ratings and number of executions for these schedules;
5) production of vectors representing users;
6) computation of distance between users;
7) choose the closest users;
8) sort results in an increasing order;
9) deliver recommendations.

We should note that among the proposed schedules, we also calculate the "currently most popular schedule" by monitoring sensors and schedules currently applied by other apartments. This solution works better in cases where there are not sufficient data for a user such as his profile or previous schedules he applied and scored or when there are not existing schedules for the specific weather conditions.

5.2.3 *Interfaces and Integration*

The Recommendation Service exposes a RESTful API, which allows the integration with other tools and services. The API manages the users' requests for recommendations and the connection with data sources, and handles user generated data. More specifically, through the API, some of the actions handled are:

- A user requests a recommendation
- A user provides feedback after applying a received recommendation
- A new user is registered
- The information about a user is updated or deleted
- Sensors to the data source flows are connected
- Open data (for example, regarding weather conditions) are added/updated
- Historical data (for example, regarding weather) conditions are received
- Users requests to retrieve all recommendations they have received and the corresponding feedback they have provided

5.2.4 *Performance, Evaluation and Stress-tests*

It is important for a recommendation service to maintain a reasonable response time and particularly the suggestion calculation to have a limited running time. It is very important, for large scale IoT deployments, especially in the context of smart city, to allow for an increased level of user experience. We have extensively tested the efficiency of our system for increasing number of users, data and complex queries and it returns high speed results. Exploiting the features of graph databases, it achieves for the big majority of queries responses in less than 100 msec. In order to achieve real-time operation features in a highly demanding smart city application domain, the suggested solution has been deployed in a high availability cluster environment, as the performance measurements have to be taken in realistic conditions, especially as the dynamic recommendation service, depending on time constraints, has to offer a user friendly experience to the user in terms of high availability and performance issue.

In order to achieve this, the Neo4j enterprise edition 3.2.3 has been selected because it allows the clustering approach. In the Neo4j High Availability (HA) architecture, the cluster is typically fronted by a load balancer HAProxy[44]. HAProxy has to be configured with two open ports, one for routing write operations to the master and one for load balancing read operations over slaves. Each application will have two driver instances, one connected to the master port for performing writes and one connected to the slave port for performing reads. The implemented cluster comprised of one server node acting as master node and 5 slave virtual machine nodes.

The goal of our distributed implementation is to provide high throughput. Each node processes a subset of the overall queries, ensuring scalability and performance in high-demand situations, reducing latency and providing continuous availability, even after a failure occurs to a machine. The proposed approach has been evaluated on the basis of read queries as to be able to determine the validity in demanding smart city applications with thousands of concurrent users requesting information. As depicted in the Figure 26, the relationship between the response time of a single server implementation and the number of users/requests is almost linear.

[44] www.haproxy.org

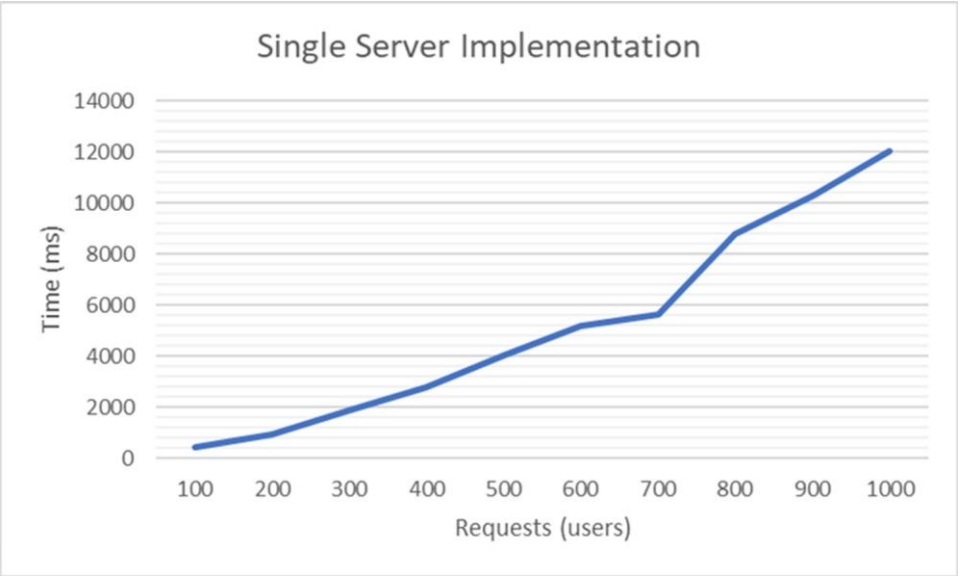

FIGURE 26: EXAMPLE OF 100 TO1000 REQUESTS – RESPONSE TIME ON A SINGLE SERVER

In order to test the scalability of our system, we examined its response time for an increasing number of requests from 100 to 1000. The process time is significantly reduced as the requests are allocated to more servers. In the following indicative figure, the total process time of 1000 requests is reduced from 12.02 seconds to 2.18 seconds after increasing the size of our cluster (Figure 27).

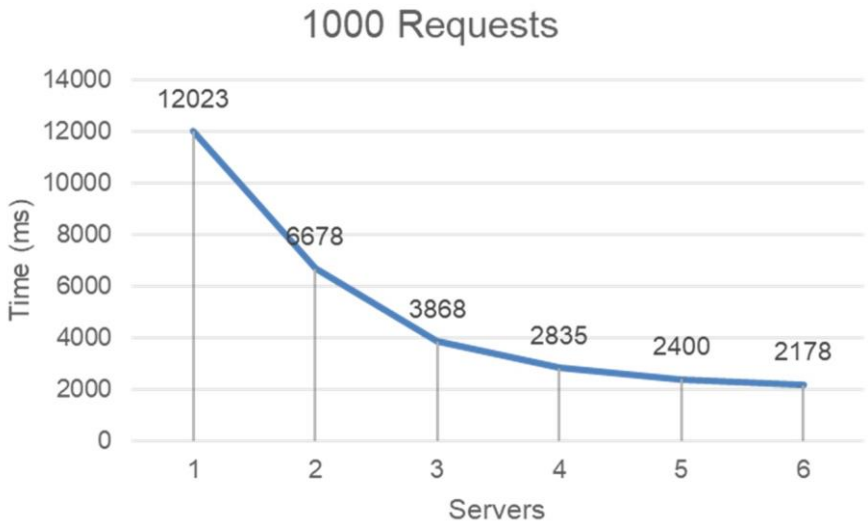

FIGURE 27: EXAMPLE OF 1000 REQUESTS – RESPONSE TIME ON UP TO 6 SERVERS-CLUSTER IMPLEMENTATION ON A HIGH AVALAIBILITY NEO4J CLUSTER

We have extensively evaluated the performance of our Recommendation Service based on High Availability Neo4j cluster in different settings. In the following representative diagram, we can see the reduction of total response time as we add more nodes (Figure 28).

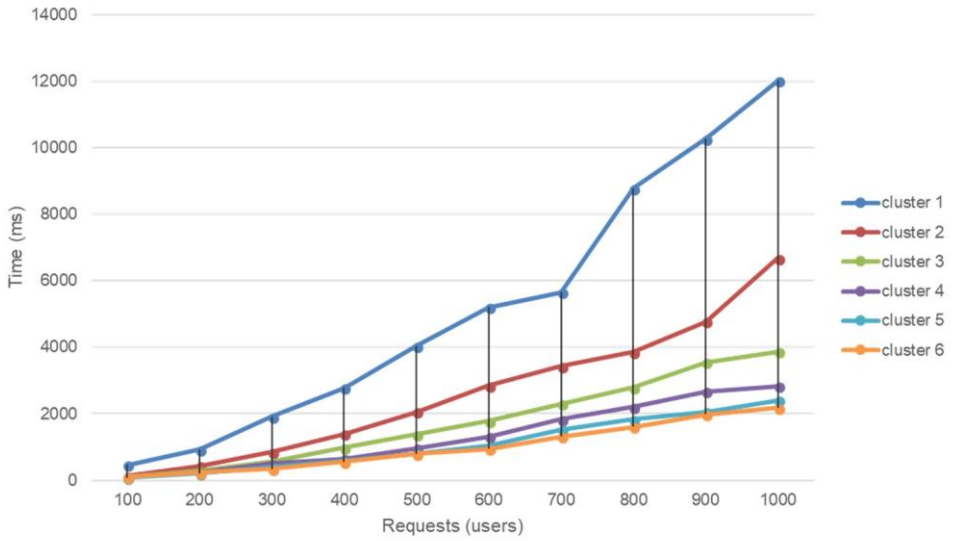

FIGURE 28: EXAMPLE OF 100 TO 1000 REQUESTS – RESPONSE TIME ON VARIETY OF CLUSTER NODE IMPLEMENTATIONS ON HIGH AVAILABILITY NEO4J CLUSTER IMPLEMENTATION

The overall improved performance of the High Availability Neo4j cluster compared to the performance of the single node Neo4j was improved in all cases. Even for experiments with a small number of requests (e.g. 300), the total response time was halved after adding one more node to the cluster (Figure 29).

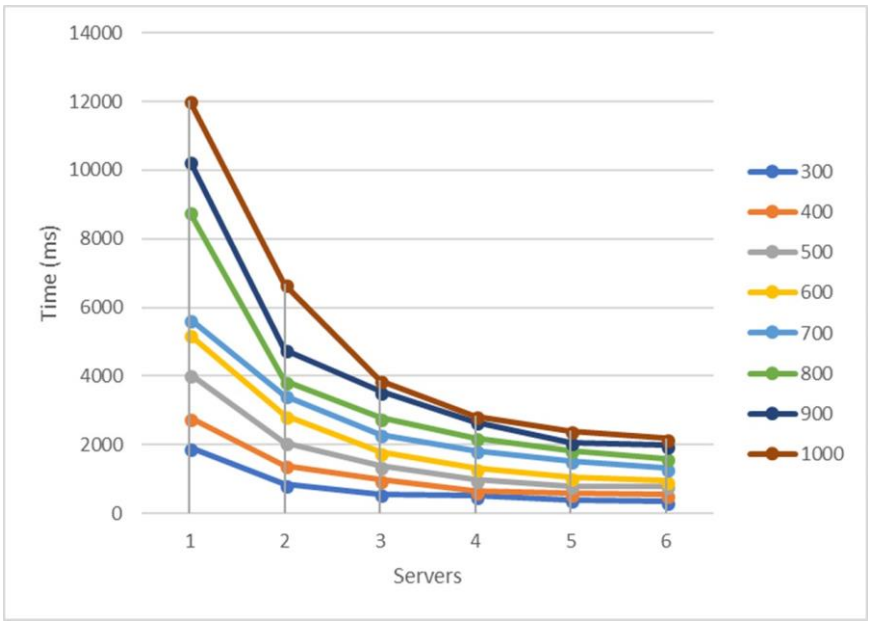

FIGURE 29: EXAMPLE OF 300 TO 1000 REQUESTS – RESPONSE TIME ON UP TO 6 SERVERS-CLUSTER IMPLEMENTATION ON A HIGH AVAILABILITY NEO4J CLUSTER

As demonstrated previously, adding more nodes to the cluster improves the performance and reduces the overall response time. However, the reduction percentage is reduced, while we add more nodes. For example, in the case of 400 queries, the transition from single server to a two-nodes High Availability (HA) cluster reduces the overall required time by 49.8%, whereas the transition from a cluster with five nodes to six reduces the required time by 5.1% (Figure 30).

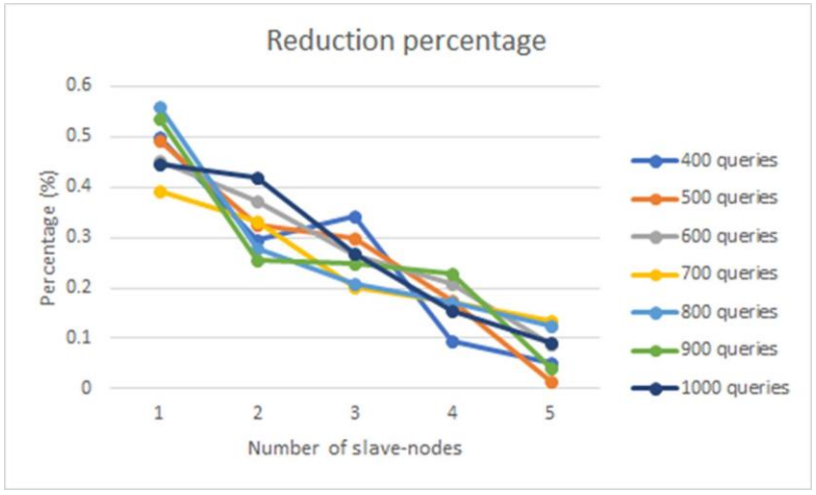

FIGURE 30: TIME REDUCTION PERCENTAGE ON A CONFIGURATION OF 400 TO1000 REQUESTS ON UP TO 6 SERVERS-CLUSTER IMPLEMENTATION ON A HIGH AVAILABILITY NEO4J CLUSTER

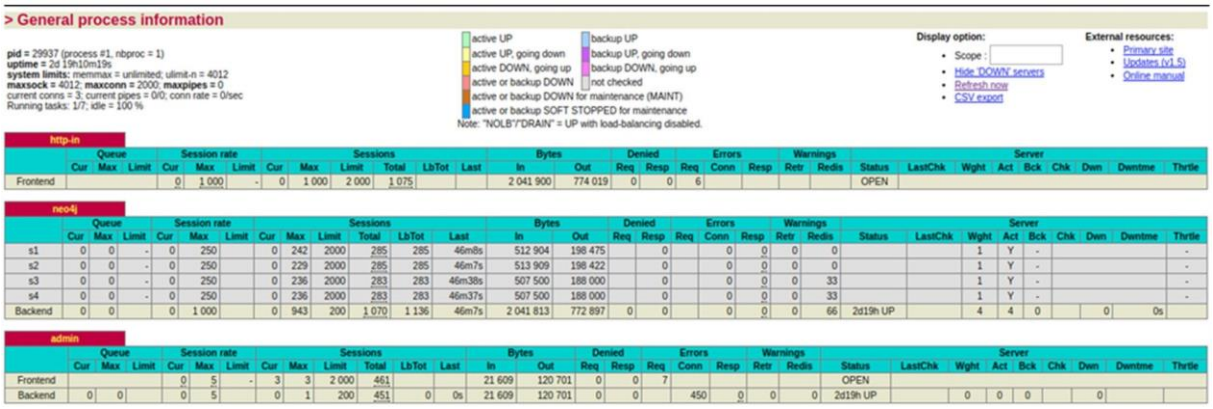

FIGURE 31: HAPROXY DISTRIBUTES REQUESTS ACROSS MULTIPLE NEO4J SERVERS AIMING TO OPTIMIZE RESOURCE USE, MAXIMIZE THROUGHPUT, MINIMIZE RESPONSE TIME AND AVOID OVERLOAD OF ANY SINGLE RESOURCE

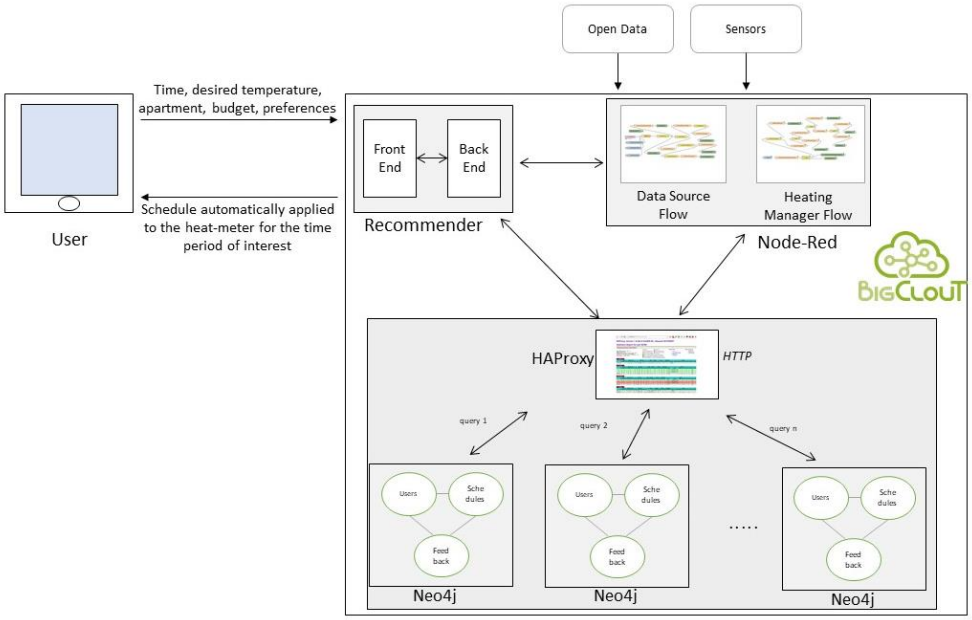

FIGURE 32: UPDATED ARCHITECTURE OVERVIEW INCLUDING HAPROXY AND DISTRIBUTED DATABASE

5.2.5 *Candidate Use Cases to be supported*

Table 1 shows in detail what functionalities of the service provided could be used in the several identified use cases.

TABLE 5: CANDIDATE FUNCTIONALITIES USED FOR EACH USE CASE

Use Case	Functionalities
Grenoble Use Case 1: Monitoring of economic impacts of events	Detecting events of interest
	Providing recommendations
	Offering predictive analysis
Grenoble Use Case 2: Monitoring of Industrial Estates	Providing information based on smart city data (e.g. public transportation) and recommend
	Providing recommendations (e.g. interesting activities, restaurants etc.)
	Detecting events of interest
Bristol Use Case 1: Smart Energy - predictive analysis of users' power consumption	Offering predictive analysis over data from sensors and open data
	Providing recommendations: the application can deliver a heating schedule for the users of the application (e.g. residents of houses).
Bristol Use Case 2: Smart Mobility	Predictive analytics to provide better paths to citizens.
	Distributed recommendation service can exploit sensor data, smart city data and other sources, in order to deliver a path.
	Distributed recommendation service provides a path with the corresponding information that a user could follow.
Tsukuba Use Case 1: Provide tourism, traffic and environmental information in real time to visitors	Predictive Analysis over data from sensors, open data and data from users (tourists).
	Recommendation application delivers an activity, restaurant etc. for the users of the application (tourists).
Tsukuba Use Case 2: Grasp status about foreign visitors to Tsukuba and provide concierge service	Predictive analysis.
	Recommendations based on collected data about foreign visitors, their profile, behaviour, histories etc. and smart city data.

6 Visualisation

6.1 General Description

Data Visualisation plays an important role in decision-making processes, allowing the user to visually represent the data and results of performed analysis. The Visualisation module is aiming to provide a tool to allow the user visualise the analysis results. This tool will cover the aspects related to the visualisation of analysis results, context information, recommendations, etc. and the production of graphical reports such as charts and maps to visualise information. In this context, the technological BigClouT asset that will provide such functionalities is KNOWAGE.

6.2 KNOWAGE

This section covers KNOWAGE's visualisation functionalities. Visualisation, in KNOWAGE's context, is related to the definition and the execution of Analytical Documents. KNOWAGE provides different types of Analytical Documents starting from a simple chart to a more complete and complex dashboard, the so-called Cockpit. Analytical Document groups under a common concept the different types of documents that can be developed with KNOWAGE when performing an analysis. Analytical Documents are used to visualise KNOWAGE's datasets and to ease the user to visually filter and analyse their information. Some specific examples of Analytical Documents are:

- **Location Intelligence Document:** which gives the chance to perform geospatial analysis.
- **KPI Document:** which permits the visualisation and the analysis of KPIs.
- **Cockpit Document:** which is used to build complex dashboard.

Every document has its own peculiarities and must be created and configured following a specific wizard. Section 6.2.1 describes the features of the Cockpit Document. For additional details about Analytical Document, please refer to KNOWAGE CE's manual[45].

As previously described in Section 4.2, the visualisation functionalities described here refer to the KNOWAGE CE version of the suite.

6.2.1 Description of Functionalities

This section is intended to give a general overview of the Cockpit document describing its main features and the widget it provides.

Cockpit

With a cockpit a user can build complex visualisation of data by simply creating and configuring different widgets. In order to start building the cockpit the user should first set as data source for the cockpit one or more datasets, manage the association among the datasets fields, if needed, and eventually configure the general style of the cockpit and its widgets. It is important to underline that every widget style property can be further customised and that every widget can use a subset of the datasets loaded into the cockpit.

KNOWAGE provides the following widgets:

- **Text:** through this widget a user can add text information to the cockpit.

[45] https://download.forge.ow2.org/knowage/Knowage_6.x_CE_Manual.pdf

- **Image:** which is used to add images to the cockpit.
- **Table:** which creates a table in the cockpit.
- **Cross Table:** which is used to create cross or pivot tables.
- **HTML:** with this widget the user can add custom HTML code to the cockpit.
- **Document:** which is used to add previously created documents to the cockpit.
- **Chart:** which is used to add several charts type into the document.
- **Map:** with this widget dataset containing spatial attribute can be easily displayed in a map; this widget is currently in an early stage and it will be released in next versions of KNOWAGE CE.
- **Selector:** which helps creating filters using directly the fields of the used datasets. Using this approach, the user doesn't have to define Analytical Drivers.
- **Active Selections:** which shows the current active selections on the cockpit.

Deepening on chart widgets, KNOWAGE provides several charts (Figure 33), in particular: Bar, Chord (Figure 34), Gauge, Heatmap, Line, Parallel, Pie, Radar (Figure 35), Scatter, Sunburst, Treemap and Wordcloud (Figure 36). Every chart can be customised following the specific needs of the analysis.

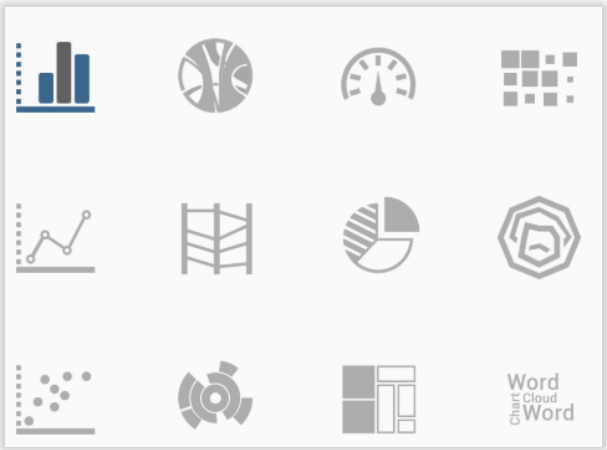

FIGURE 33: KNOWAGE CHARTS TYPES

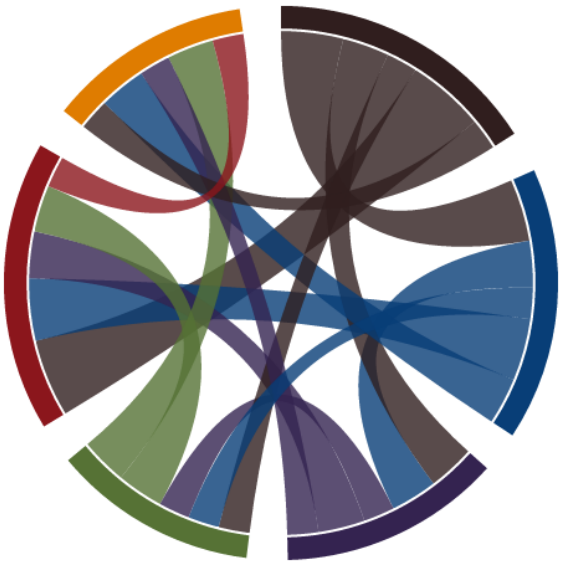

FIGURE 34: KNOWAGE CHORD CHART EXAMPLE

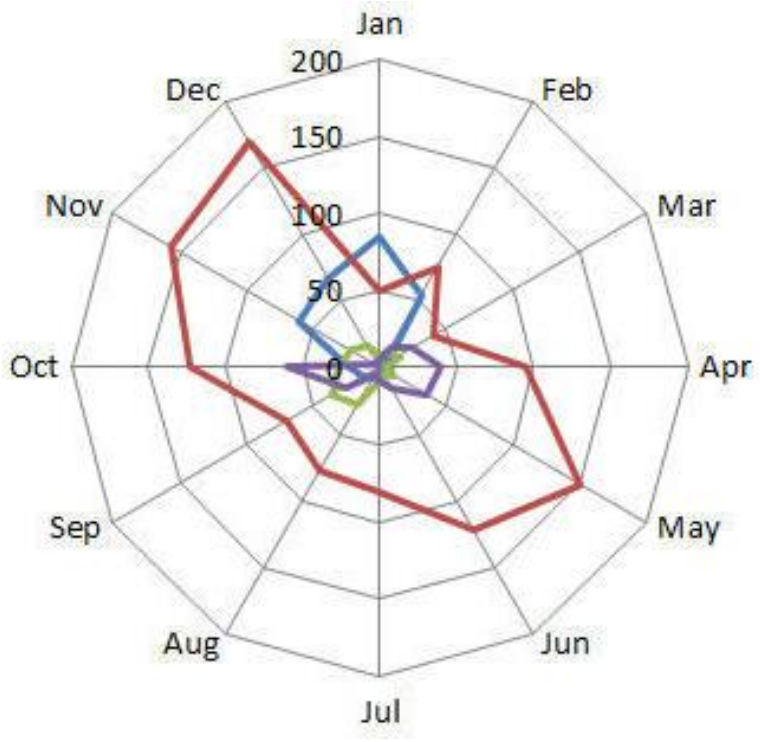

FIGURE 35: KNOWAGE RADAR CHART EXAMPLE

FIGURE 36: KNOWAGE WORDCLOUD CHART EXAMPLE

It is important to underline that a cockpit can be composed of one or more sheet, and each sheet can contain different widget; in this sense cockpit is a multi-sheet, as depicted in Figure 37. Moreover, it is also an interactive document, and every single widget will be automatically updated by clicking other widget element, if an association among the widgets' dataset is provided.

FIGURE 37: KNOWAGE MULTI-SHEET COCKPIT EXAMPLE

6.2.2 Implementation details and Internal Architecture

As described in Section 4.2.2, KNOWAGE's architecture (Figure 15) is composed by three layers: Delivery layer, Analytical layer and Data layer. Visualization functionalities of KNOWAGE takes advantage of the Analytical and the Delivery layers in order to produce and visualise analysis through charts and cockpits. The Analytical layer provides analytical features and capabilities easing the production of charts and cockpit, moreover it manages the visibility of documents and dataset according to the user's role. The Delivery layer guarantees data access and visualisation through KNOWAGE's GUI or RESTful APIs.

The framework used to produce charts is Highcharts JS[46] or Chart.js[47], depending on the library chosen during the installation (for more details please refer to document D3.3 Big Data Analytics Framework Prototype – Demonstration). Resulting charts could differ depending on the library.

6.2.3 Interfaces and Integration

KNOWAGE's GUI allows the creation of documents, such as charts and cockpits, through an intuitive and interactive interface managing widgets configuration and style properties. Moreover, KNOWAGE provides REST APIs that an external application can invoke in order to create or list documents. The specific APIs used to manage the Document resource are:

- **List All Documents:** which returns the full list of the available documents.
- **Adds a new document:** which adds new a document. In order to add a new document through this API, a user must be aware that any dataset or data source linked to the document must be previously created.
- **Return document with specified label:** which returns the detail of the specific document.
- **Update the document:** which allows to update a document.
- **Delete the document:** which deletes the document.

The full list of the available APIs and their details are accessible in a dedicated Apiary.[48]

[46] https://www.highcharts.com/
[47] https://www.chartjs.org/
[48] https://knowage.docs.apiary.io

6.2.4 Compliance tests

Tests about Visualisation of data are performed taking advantage of the data gathered through BigClout's CKAN repository. Indeed, by using garbage truck Pm2.5 measurement in the city of Fujisawa, several cockpits are produced showing the average Pm2.5 measures per day or the time trend of the measures giving the chance to the user to select one or more device and a specific day of interest. These measures are displayed combining different widgets, in particular: maps, charts and tables.

Figure 38 illustrates an example of Cockpit that shows the average Pm2.5 measurement retrieved through the garbage trucks in the city of Fujisawa. In the map widget a HeatMap is used to show the measurements in the city. The bar chart on the right of the map shows the average values per sensor. Moreover, the selector widgets at the bottom of the bar chart will help the user filtering the analysis.

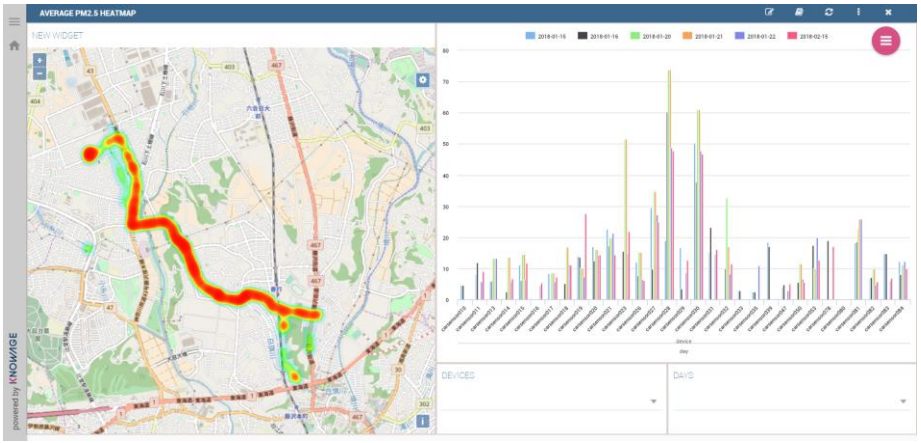

FIGURE 38: AVERAGE PM2.5 COCKPIT

Figure 39 depicts the measurements of a single garbage truck in a single day. In particular, the map widget shows the punctual measures in the city. The line chart displays a relation between the Pm2.5, the blue line, and the speed of the vehicle, the black line and area. In the table below the user will see a summary of the different measurements. One of the main features of this Cockpit is the usage of the Analytical Driver that models the devices and the days of the measurements. These drivers are displayed in the right section of the Cockpit.

FIGURE 39: SINGLE DEVICE MEASUREMENTS WITH ANALYTICAL DRIVER

6.2.5 Candidate Use Cases to be supported

This module offers specific tool that starting from analysis results can visualise the data in several manners. Thanks to its generic behaviour it will be able to construct graphical representation of most of the data produced by other tools of BigClouT's platform. It requires that data must be accessible by KNOWAGE. From the Use Cases presented in D1.4 and their requirements this component could support:

- Fujisawa's "*Use Case 1: Optimizing the incidence on local economy of Fujisawa"* by visualising the produced data.
- Fujisawa's "*Use Case 2: Fine-grained city infrastructure management"* by visualising the analysed data.
- Bristol's *"Use Case 1: Smart Energy - predictive analysis of users' power consumption"* creating dashboards to show the produced information to users.
- Bristol's "*Use Case 2: Smart Mobility"* visualising green or red paths in a dashboard.

7 Integration plan

To facilitate the integration and demonstration of the various components (of WP3 and BigClouT in general), it is necessary:

i. for the **technical partners** to find ways to connect their tools with each other by finding common technological ground: to that end, for each component we presented a description of functionalities, implementation details as well as interfaces and integration details.
ii. for the **technical and pilot partners** to find common application ground: to that end, for each component we presented a description of functionalities, interfaces details, compliance tests results (linked to use case requirements) as well as candidate use case scenarios.

Coupled with the technical details provided in D3.3 (Package information and Installation instructions, User Manual, Demonstration – Demonstrator, Licensing information), this document can be used as a roadmap for the implementation of the overall integration plan.

More specifically, due to the presentation of various candidate use cases to be supported by the WP3 functionalities, it is now easier to map all services to specific demonstration scenarios. Moreover, since the technical details of these services are presented in depth, it is possible to combine most of (if not all) the services mapped at these same scenarios.

8 Conclusions

This deliverable presents in great depth the Big Data Analytics Framework of BigClouT project and its architecture. Due to the detailed and structured presentation of the several elementary services and functionalities WP3 modules and components can provide, this report can be used during the third year of the project as a reference document and a roadmap for future actions regarding integration and (integrated) demonstrations.

The input provided by all WP3 partners in both D3.2 and D3.3 will be used to define and finalise the implementation, integration and demonstration plan of the Big Data Analytics Framework that will be responsible to provide these new functionalities in BigClouT.

Information provided in this document, and in general results produced by WP3, in terms of use cases, sequence diagrams and technical details about integration of the different assets composing the *City Data Processing* will be taken in input by other work packages in order to continue the work towards the final results of BigClouT.