Linked Open Data Management in Ambient Assisted Cities

Ruben Mulero
DeustoTech Institute of Technology
University of Deusto
Bilbao 48007, Vizcaya, Spain
Email: ruben.mulero@deusto.es

Vladimir Urosevic
Belit Ltd.
Obilicev venac 18-20,
11000 Belgrade, Serbia
Email: vladimir.urosevic@belit.co.rs

Aitor Almeida
DeustoTech Institute of Technology
University of Deusto
Bilbao 48007, Vizcaya, Spain
Email: aitor.almeida@deusto.es

***Abstract*—Linked Open Data in smart cities are an emerging source of open information comprising the needs and requirements of the citizens. However, there are some limitations when it is necessary to store data in sets of different abstraction and aggregation levels. In this paper, we present a new approach to provide smart cities with a tool for storing the data from citizens that provides the ability to select the type of abstraction or aggregation level required in each moment. In addition, the paper presents a novel way to share semantic data following the Linked Open Data paradigm by using a rule engine based reasoner inferring new statements based on spatio-temporal rules.**

I. Introduction

As a result of the growth of urban population worldwide [1], their positions as one of the central elements in human organization. The concentration of resources and structures around cities offer new opportunities to be exploited. *Smart Cities* [2] [3] are emerging as a paradigm to take advantage of these opportunities to improve their citizens lives. The City4Age[1] platform is a H2020 research and innovation project with the aim of enabling age-friendly cities. The project aims to create an innovative framework on ICT tools and services that can be deployed by European cities in order to: enhance early detection of risk related to frailty and Mild Cognitive Impairments (MCI), and provide personalised intervention that can help the elderly population to improve their daily life and also promote positive behaviour changes.

As part of the tools created for the framework, we have developed a city-wide context-data management system that serves as a central information repository for the project. In the past, we have worked on developing platforms to manage the context [4] in intelligent environments [5] [6] [7], but limiting ourselves to a single home or building. In the case of City4Age, the created context-manager is able to simultaneously manage and integrate the information produced by several cities, while being flexible enough to allow each city to decide the level of abstraction of the provided data.

The objectives of the City4Age context-manager are twofold. The first one is to provide a data repository for all the information generated in the cities (i.e. activities of daily living, behaviour patterns, detected MCI and frailty risks, proposed interventions, etc.), integrating it and allowing for the analytic algorithms that use that data to access it in a fast and timely manner. The second one is to provide a mechanism to give semantic meaning to the stored information and to share it with third parties while preserving the user privacy. To tackle this problem we have designed an architecture with two main elements: 1) A high performance REST application programming interface (API) that allows to manage large quantities of data easily and 2) a Linked Open Data (LOD) API that maps the information in the database to Ontology Web Languague (OWL) [8], providing semantic meaning to the stored data and making it easier to share. Following the Linked Open Data [9] paradigm we ensure that the provided data will be easily understandable and usable by third-parties, being them humans or machines.

This paper has the following structure: Section 2 reviews previous work related with context management and *smart cities*. Section 3 introduces the architecture and characteristics of the City4Age city-wide context manager. Section 4 describes the evaluation of the system. Finally, Section 5 draws some conclusions and outlines future work.

II. Related work

In the last years, researches have combined different technologies to give to the cities the concept of 'smart'. The inclusion of Internet of things (IoT) and the citizens active participation inside their environment are opening a new way to study the impact of their behaviours in different situations. Citizens are developing a new set of requirements in the daily live, hence it is necessary to research new ways of city management to satisfy their requirements and know exactly what they may need. With these requirements in mind, the concept or *smart cities* appears in the literature [10] [11] [12] to try to help citizens in different areas such as health, administration, energy or transportation.

Smart cities are the natural evolution of our actual society, not only because more humans are living inside a city but also because the connection between the humans and the cities are more closer than ever and in some areas this can help to build solid infrastructures to satisfy the citizens demands [13]. The adoption of Internet of Things (IoT) and the creation of nodes to connect smartphones, smartbands, movements sensors, cameras and all new technologies involving with the Internet are

[1] http://www.city4ageproject.eu/

opening new capabilities to connect humans directly with their city and its environment [14] and provide useful services to satisfy automatically some basic requirements. In addition, it opens a new gate to gather users data an analyse them to obtain useful information and extract behavioural patterns, detect potential health risks or study their level of education to adapt some services to their comprehension.

In the recent years, the conception of Linked Open Data and *smart cities* are establishing a new way of obtaining source of data to share knowledge between computers and create different services [15]. The cities play a pivotal role to extract data directly from citizens and cover a wide range of areas with their personal information. Hence, a new research area opened and various different projects have been launched to study the impact of the *smart cities* in the citizens lives while they gathers data to be shared open over the Web. Some approaches involve the extraction of data through IoT devices and manage cognitive information throught a framework that stores data in Resource Description Framework (RDF) [16], other approaches create semantic frameworks to annotate streaming sensors data by using the semantic web to share geographic positions in real time [17]. If the source of data is a *smart city* it is possible to create *smart city data* to gather semantic data from citizens and create an API to connect different applications [18] and provide useful information.

There are another approaches such as the creation of a cloud of things by combining different IoT platforms, cloud computing and semantic data to combine different IoT middleware and have a visualization tool of gathered data [19]; or a platform to design an Ontology based on a convertion of SPARQL Protocol and RDF Query Language (SPARQL) queries into Structured Query Language (SQL) queries to gather data from relational database with a semantic meaning by using some rules as a primary conversion tool [20].

In all different approaches the IoT technology acts as a primary source of data and Linked Open Data is the final tool to represent that data. In general terms, the majority of the solutions try to construct a new middleware or a combination of middlewares to extract data through IoT devices and share it using the Semantic Web to find an innovative way of data representation. Some of them, uses rules to try to map some semantic queries into relational queries but without giving to semantic data additional meaning, others try to create a useful API to connect different applications and use semantic data for different purposes but without expanding data or creating new semantic statements. In all situations, none of them try to apply a reasoner to generate new knowledge and expand it into the context of the *smart cities*.

Knowing that the relational databases are the primary option to store data in the majority of the projects we decide to develop our approach based on this requirement and create a system that can obtain data through an IoT middleware to store it into a relational database. Then, by using this source of data, we give it a semantic meaning to use a rule engine reasoner and infer new statements based on a spatio-temporal rules. Our approach defines a new tool to reuse stored data in a relational database and load and expand the knowledge into a desired direction based on the needs of each city context.

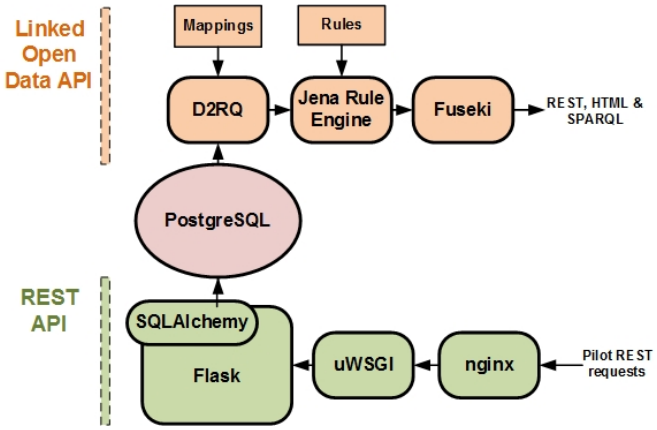

Fig. 1. Overview of the complete architecture

III. System Architecture

As mentioned in the introduction, the system architecture is divided in two parts. The first part, contains a REST API that manages large quantities of data. The second part, contains the LOD API to read and creates semantic data from a relational database using a preconfigured mapped file. The source code of both implemented solutions is available on Github [21].

Figure 1 depicts the overall architecture of the implemented solution. As it is possible to see, there are different technical technologies involving in, the majority are well-know but there are some of them that requires a further description due to the complexity that they have. The descriptions of each of them will be detailed later.

As it is depicted in Figure 1, the Pilots (Cities) send REST requests to the API to send their gathered data. These Pilots obtains their data by using a sensing middle-ware which uses different type of technologies to collect citizens' data, such as Ambient Living Display when they are in home, smartphones, wereables, movement sensors, cameras, low energy bluethood beacons and so on. This middle-ware manages data from its sources and sends it to the REST API. Then the REST API, handles the data and stores it securely into a relational database. The source of data comes from different Pilots cities from the City4Age project. The Pilots send the data to the REST API about citizen actions to store some behavioural patterns and detect, for example, if they have frailty or mild cognitive impairment. The implemented solution is prepared to store their personal data in a secured silo with a range of different secure measurements, such as encryption, hashing and token web access.

Once the data is stored into the database, The LOD API obtains the data and gives it a semantic meaning. This resulted data has a common sense and meaning that can be read both by humans and machines, hence this data can be called as *knowledge*. With *knowledge*, it is possible to use a rule engine based reasoner to infer new statements and expand it to a new

```
{
  "action": "eu:c4a:usermotility:enter_bus",
  "user": "eu:c4a:pilot:lecce:user:12345",
  "location": "it:puglia:lecce:bus:39",
  "payload": {
     "position": "urn:ogc:def:crs:EPSG:6.6:4326",
     "sensorId": "sensorId:beacon_123456"
  },
  "timestamp": "2017-01-22 07:08:41.22222",
  "rating": 0.4,
  "extra": {
     "pilot": "lecce"
  },
  "secret": "jwt_token_k"
}
```


Fig. 2. JSON example code

direction according to the needs of the citizens. These new expanded knowledge can be shared by using a service that provides different endpoints to allow potential users, consume the data.

All data transactions are encoded in *JavaScript Object Notation* (JSON) and they need to follow various guidelines to be accepted by the REST API. These guidelines, derived mainly from the Web of Topics (WoX) conceptual model approach [22], help to create uniform transfers and establish a standard for each type of data that it is necessary to store into the database. Figure 2 depicts a sample code in JSON to store an action performed by a citizen. The sample contains data information about the location of the user, the type of sensor which detects the action, the uncertainty of detection (in percentage), the exact position of performed action and the name/identifier of the action (enter the bus). Additionally, there are some extra fields to add a security layer to the data and there is an 'extra' information field to send to database if the action or devices needs to clarify some special situation.

The reason behind the use of a relational database rather than a triple-store database [23] is because relational database is faster and powerful. In addition, relational databases are more commonly used in the majority of application solutions such as *machine learning* thus, this ensures compatibility and scalability to expand the concept of this solution.

A. REST application programming interface

The aim of the REST API is to provide an on-line solution to manage data from different sources and store it in an secure environment. The implemented architecture defines a mechanism to insert data in a different logical levels. Hence, it makes possible to define what level of abstraction needs each city to manage citizens data. The API leaves to the cities decide what kind of information needs to be stored into database. There are various types of abstraction levels, each of them defines the precision of what kind of data is going to be saved. On the first level the example method is *add_action*, storing the data that describes an elementary action performed by a citizen (e.g. enter_bus), usually captured by a single IoT sensory device. This endpoint represents the lowest abstraction level handling mostly raw or calculated technically measured data in a common structure. The second, exemplified by the *add_activity* method, stores the activities performed by a citizen, mostly collections or sequences of elementary actions that constitute an activity (visit_cousins for example). The primary aim of this endpoint is an abstraction level providing data representation of defined or recognized behavioural activities performed by a citizen. The third, with the *add_measure* example method, allows to store geriatric/medical measures of behaviour variation of citizens. Measures are represented through a further grained hierarchical structure with factors provided by geriatric and behavioural experts. Each factor has a set of subfactors to represent the aspects of the measure sent to the REST API. For example, a factor could be called "motility" with a subfactor of "walking", this combination has a different kind of data measures attached to this situation.

A visual example of how is structured a JSON based *add_action* entry is shown in Figure 2. This example represent the low level information that can send a city to the REST API. The information given in the JSON will be stored in a defined database schema that will be described later.

The implementation of the proposed solution is based in Python programming language but other type of technologies are used. The API is defined mainly by two technologies: 1) *Flask microframework* and 2) *uWSGI application container*.

1) Flask micro-framework: The Flask micro-framework [24] is a Python library which helps in he creation of Web Services. The motivation is to create a simple Python program which can solve various requirements. These requirements are fulfilled by using an endpoint with a coded method.

When the configuration of Flask is completed and the endpoints are properly coded with each needed service, the Web Service is started as a Python main program and Flask does all the work. Flask relies on Werkzeug, an implementation of Web Server Gateway Interface (WSGI) [25] protocol, to have a set of libraries which helps in the manipulation of Web services and Web Servers. The reason to use this type of framework is because it is simple, secure and it only needs a few Python programming language knowledge to create full customizable Web Services.

2) uWSGI application container: uWSGI [26] is an another implementation of WSGI protocol. The difference between uWSGI and Flask is that uWSGI works as a application container and creates an abstraction between the implemented code and the Web Servers. uWSGI handles data transmissions between Python code (Web Services) and a reverse proxy based Web Server, for example, a Nginx [27] Web Server. uWSGI is needed because Web Servers are not prepare to handle Python code directly, thus it is necessary to use a middle software to execute Python interpreter to use Flask and create the API service. Figure 3 depicts the logical implementation of uWSGI.

The *"config.ini"* file is an uWSGI config tool to define the behaviour of this technology and tell it for example, where is located Python Flask application, where should be created the socket, *nix permissions for the created socket and so on.

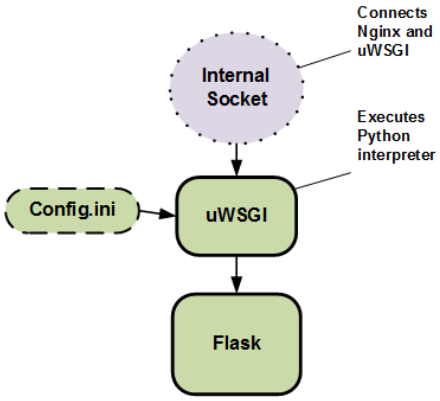

Fig. 3. uWSGI logical implementation

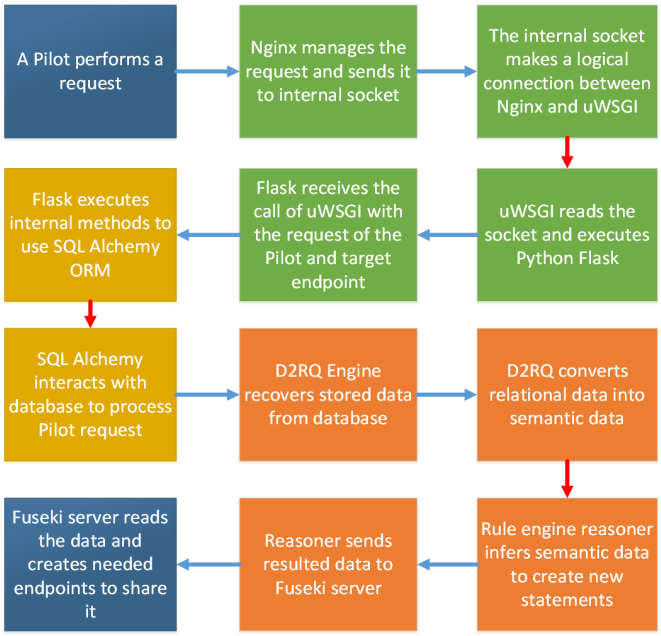

Fig. 4. Workflow of the proposed solution. The blue squares represent the input and output data. The green squares are part of the REST API workflow. The yellow squares are part of the database workflow. The orange squares are part of the Linked Open Data API workflow.

B. Linked Open Data Interface

The LOD represents the solution to give a semantic meaning to stored data and share it by using different type of endpoints. The aim is to extract data from a relational database, map it with a previously created Ontology to create semantic data and then, use a rule engine reasoner to infer new statements based on a certain rules to expand the knowledge to a desired direction. The rules need to represent the semantic context of the *smart cities* to fulfil the requirements of the citizens and create useful data. Figure 4 gives an overall view of the implemented solution.

This part uses the same database as API but, here the data is modelled with the help of an external program that uses a map file to fit every table and column of database with a designed Ontology. As it is possible to see in the Figure 1, data is modelled from database to the final output.

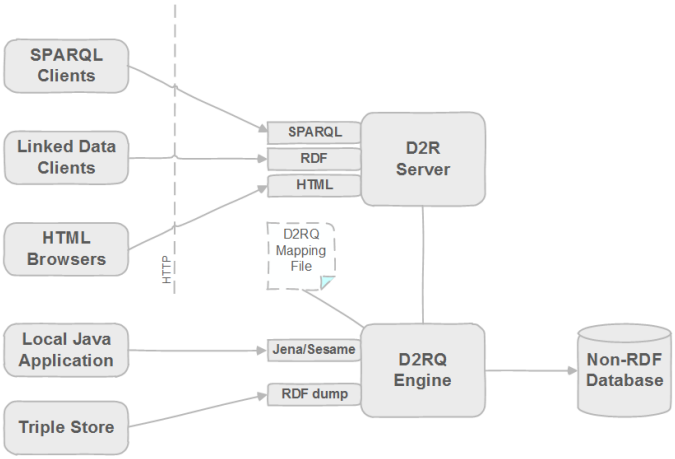

Fig. 5. D2RQ logical functionality

This solution uses three technical tools: 1) the mapping of the data is done by *D2RQ* platform; 2) the rule engine inference program is done by *Jena* libraries and Java programming language; 3) the endpoints to access to the final data are served by *Fuseki*.

1) D2RQ Platform: *D2RQ* platform is a set of tools written in Java programming language for accessing relational databases as virtual read-only RDF graphs. This technical tool provides a solution to extract data from relational database and share it by using a variation of Pubby server [2] called D2R server. Figure 5 depicts logical implementation of D2RQ platform.

In the implemented solution, D2RQ plays two roles. Firstly, the extraction of relational data by using a mapping file (D2RQ Mapping file) to attach the data to a previously designed Ontology. This mapping file has a self programming language called D2RQ mapping language [28] to connect directly to the target database and append each table and column with the classes and properties of the designed Ontology. Secondly, the use of its API based on *Jena* libraries (will described later) to extract and manage the loaded knowledge to use a rule engine reasoner and infer new statements.

2) Jena: *Apache Jena* is a framework to build applications based on Semantic Web and Linked Data. The primary aim of this framework is help to developers manage semantic graphs. Additionally, these libraries comes with an inference support based on a four different type of rule engine reasoners to infer some knowledge and create new statements.

From Jena, we used its generic rule engine reasoner with a rule file based on a set of forward rules to fulfil the requirements of the citizens. These requirements are movements made in a defined place, or some typical actions in their house like 'put the milk in a bowl' when the citizen want to take the breakfast in the morning. These typical actions can be inferred according to the time, type of action and the place which has been performed.

[2]http://wifo5-03.informatik.uni-mannheim.de/pubby/

3) Fuseki: Fuseki is a SPARQL server. It provides REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the SPARQL protocol over HTTP. The reason to use Fuseki to serve Linked Data is because the implementation of D2RQ server (*D2R server*) is only designed to be as a read only server. In our approach, we are loading data and generating new knowledge, thus it is necessary to use an external tool to share this new data due to the limitations of D2R server.

The Fuseki server comes with a tool called Shiro to create some protection rules. This tool allows to configure some security measures to have a control of the loaded data and the users who can access to the server.

C. Operation logic

The operation logic of this system is represented in Figure 1. The functionality comes from bottom to upper. The initial requirement of this system is to have a previously designed Ontology which represents the context of the citizens and their life inside in a *smart cities*. We used the City4Age project Ontology to have the needed representation of semantic data. Using this Ontology we create the file called *mapping file*. The mapping file is used to access into relational database and extract the data according to the definitions written in the file. In addition, we create a set of rules to infer new statements when the rule engine reasoner is used, following an approach that we used previously in [29]. These rules needs to represent some logical and comprehensive reasons to expand knowledge and represent better the semantic data. An example of two created rules are exemplified in Table I. Both rules are an example approach to understand how it works a rule in the implemented rule engine reasoner.

The first rule checks the locations of loaded data and detects if each location has an associated pilot and if it is registered in an indoor place. If these requirements are satisfied, then the rule is executed and it creates a new statement to represent that the subject is inside a building; combining two different Ontologies (Schema[3] and City4age). The second rule obtains the subject of an executed action (*add_action*) and checks if the registered action is between two defined dates. If the rule is satisfied then the rule engine creates a new statement giving to the subject a new meaning as a status "registered".

Once the mapped file is created and the rules are defined, a Java based program is used to perform various actions. The Java program loads the knowledge using the mapped file and saves it into the memory. The loaded knowledge is inferred by Jena Generic Rule Engine Reasoner [30] with the previously created rules file to infer new statements that will be added to this loaded knowledge. When all knowledge is prepared, the program makes an internal call directly into Fuseki's SPARQL endpoint to insert all knowledge into the server dataset . The server will serve the data providing the necessary endpoints.

One of the drawbacks of this method appears when de data in database is updated, this changes should be reflected in the semantic knowledge loaded into Fuseki server. The implemented solution to this problem is to repeat the above procedure periodically in a defined time. Using this approach the knowledge is destroyed and is loaded again with the updated data. The advantages of this solution is that we can modify the rules file to change the behaviour of the rule engine reasoner and adapt the knowledge to the requirements of the citizens without restarting the server.

[3]http://schema.org/

IV. The City4Age data model

As said in previous sections, data is stored into a relational database. This database uses a designed data model to ensure that is capable to represent the context of the *smart cities*. The cities can decide which information they want to store into database by giving them a set of different levels of abstraction. This levels are designed to let to cities decide what kind of information they want to send and store into the system. Each level contains more data requirements that needs to be satisfied in order to use it, in other words, when the level of abstraction is more deeper, the required data shall be more accurate.

The designed database contains two different schemas, each of them is designed to store different type of citizen personal data. The schemas are listed as: 1) The Activity recognition, which stores data gathered from different sensors. This schema stores the data related to the user actions and activities; 2) The Shared repository, which stores medical measures provided by expert systems and external users (geriatrics, medical researchers and so on). This schema stores the data related to the user medical measures like blood pressure, height, weight, some remarkable health problems and so on.

The logical implementation of two schemas, covers the necessity of having a REST API that can isolate behavioural data and medical data. The behavioural data is more related to psychology science whereas the medical data is more related to health science. This two logical schemas helps in the detection and gathering process of the data by other external tools that needs to interact with the database.

A. The Activity Recognition schema

The Activity Recognition schema is designed to store actions and activities performed by the citizens. The model uses a set of entities that allows to store what actions are performed in a defined time and place and what are the activities that can show these actions. The actions are represented in the *ExecutedAction* entity and it has some parameter values that exhibits the measures of the action such as the action name, the confidence ratio, the location of the action, the Pilot (city), some extra information if the action needs to be described and, optionally, an activity attached to it. The *Activity* entity represents the meaning of a collection of *ExecutedAction*, this entity is a top layer of information which represents the different type of actions that can be represented as an Activity. Activities can performed by a user in different places.

An example of the described entities could be "put the milk in a bowl" executed action that is part of "Prepare breakfast" Activity. These two entries has their measures

TABLE I
AN EXAMPLE OF RULES APPLIED IN THE TEST

Name of the rule	*"When"* declaration	Direction	*"Them"* declaration
buildinglocation:	(?subject rdf:type vocab:location), (?subject vocab:location_indoor "t"), (?subject vocab:location_pilot_id ?object)	>	(?subject schema:Place city4age:Building)
registeredstatus:	(?subject vocab:executed_action_date ?object) greaterThan(?object, "2014-04-21T07:08:41"8sd:dateTime) lessThan(?object, "2014-08-21T07:08:41"8sd:dateTime)	>	(?subject schema:actionStatus "registered")

and they give different information according to the level of required abstraction. In addition, there are more entities that store some additional parameters derived to the previously described entities, but the purpose of these entities is to make data more scalable and organized.

The design of this schema covers the necessity of store the actions performed by the citizens in a defined spatio-temporal time, thus it allows to decide if the cities need to store exact actions with a hight level of accurate (*ExecutedAction*) or if they only need to store a more complex and complete set level of activities with their each *ExecutedAction* performed in a time range.

B. The Shared repository schema

The shared repository schema stores medical conditions of the citizens provided by medical experts. This part of the model stores different kind of physiological measures of the citizens to have some base information and know exactly, if a user needs an intervention when an external expert system detects that the user could have mild impairment, frailty or a degenerative disease. The model is represented by *NumericalIndicator* entity which stores some medical measures gathered by experts, this numerical indicator represents variables with accurate information. The *DetectionVariable* entity which stores the information of what disease has been detected in a citizen and the *CareProfile* entity which stores data about blood pressure or physiological measures.

This schema gives a medical information to experts to manage the information about the citizens and try to decide if a user is suitable to suffer a degenerative disease or not. The information given to this part of the model, is generally based on experts that use a dashboard to send data to the REST API with their calculated measures levels and indicates if a defined user would need an intervention when its health is decreasing.

V. VALIDATION

To test/validate the context manager we have performed two type of test. The first one is a stress test to know if the API can afford heavy loads of data, this test ensures that the the API is capable to handle multiple client connections and save massive data from different citizens into database.

The second is a *black box test*, the primary idea is to know exactly if the implemented solution works as intended by sending a sample of data and testing if a different type of rules infers new statements. Then, the test checks if inferred statements and actual loaded knowledge is sent into Fuseki server and is available to query with SPARQL.

A. Stress tests

To perform the stress test, we decide to use JMeter [31] to simulate REST clients sending information to the API. To do the experimentation, we create a random list of different *add_action* samples as we explained in Figure 2. The resulting file contains a list of 30 random samples. To make this test more realistic as possible, we simulate 800 citizens sending information to the system in an intervals of 0.5 seconds. Each of them sends three different requests.

The first, is a GET request to the main page of the API, it returns a welcome message with a status code 200 to the user if all works as intended. The second, is a POST request where the user sends its login credentials to be authenticated inside the API, we assume that all users sends valid user login requests, thus the system will returns a request message to inform the successfully login with a status code of 200. The third, is a POST request where the user sends the list of 30 JSON instances of *add_action* to store it into database. If data is stored and API can confirm the committed action, then it will returns a 200 status code with a message to inform that everything is correct.

The the simulation environment is a dedicated server based in a Intel(R) Xeon(R) E5606 processor at 2.13GHz of clock speed, with 8GB of RAM memory at 1333Mhz clock speed (In dual channel mode) and a 500GB ATA disk with maximum transferred speed ratio at 300MB/s and 7200 nominal media rotation rate.

The total execution time took 6 minutes and 41 seconds to complete. We repeat this test five times to ensure that results where always the same in each test case. In all iterations, the stress tests was finished successfully without no errors. The results of the simulation are depicted in Figure 6.

As we can observe in the results, the implemented solution can handle multiple massive request in a short period of time. This confirms in which the API can handle requests from different citizens without no major problems. In addition, we can confirm that the API is well optimized when users need to enter new data into database. As it is depicted in Figure 6. The login request takes more time than other requests to complete, this behaviour is logical because the implemented solution creates an encrypted cookie for each successfully logged client. In the following iterations, the user will be

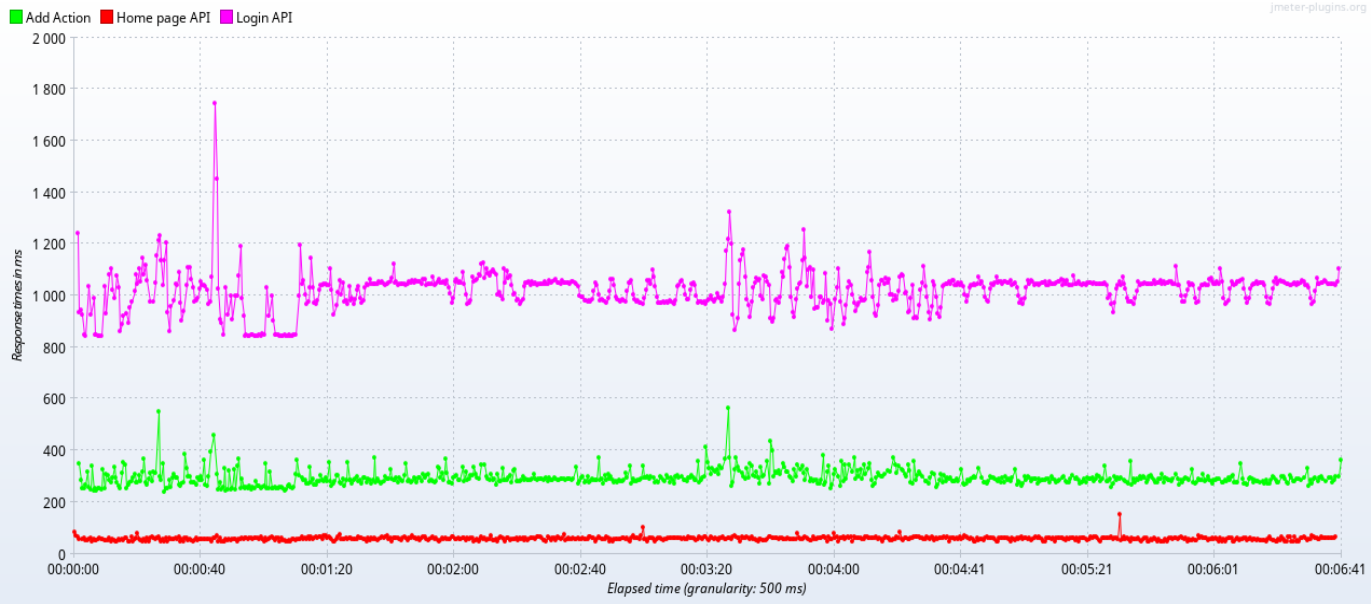

Fig. 6. Response time of API in the stress test

logged into the system, thus it will not require to login again and it only will need to add data into the database. The *Home page API* data contains the minimum execution time performed by a simple request to the APIS's home page. The differences between home page requests and add action request are minimum in terms of time response hence we can confirm that the implemented solution works as intended and it can handle massive amounts of data from citizens.

B. Black box test

The *black box* test is focused into LOD. To accomplish this part we decide to simulate the action of add new data into the system and then, study if the rule engine reasoner infers new statements based on a set of rules. To test if data is loaded and inferred into Fuseki server we send a SELECT SPARQL query to evaluate the results. If data and inferred data are loaded properly, we can confirm that the rule engine reasoner is generating new knowledge based on our set of rules and this knowledge is being sharing over the Internet. To validate the test results, we make a study of the logging information generated by the rule engine reasoner program to ensure that the data is correct.

The data used in this test are the previously exemplified rules in Table I and the *add_action* samples of Figure 2 sent in the previous stress test. The SPARQL query statement sent to the server is showed in Figure 8.

The SPARQL query results showed in Figure 8 reveals that the data is loaded into Fuseki and is being shared over the internet because the sentence returns the "them" declaration coded into the rule number two. In addition. we study the results gathered by the logs of the Rule Engine Reasoner, this results are showed in Table II. Apart from the results of the table, the program logs gives additional information

```
prefix schema: <http://schema.org/>
prefix deusto: <http://www.morelab.deusto.es/#>

SELECT ?sub ?obj WHERE {
  ?sub schema:actionStatus ?obj
}
```

Fig. 7. SPARQL sample query used in the *black box* test

```
deusto:executed_action/11
deusto:executed_action/6
deusto:executed_action/5
deusto:executed_action/12
deusto:executed_action/7
```

Fig. 8. Part of SPARQL results in *black box* test

TABLE II
RESULTS OF THE RULE ENGINE OUTPUT

Number of elements	Execution number	Inferred Count
757	1018	54

such as successfully connections to Fuseki server, a list of generated RDF graphs and an error handling, thus it is possible to confirm that the rule engine reasoner is doing the inference job correctly and it is not giving errors.

The results given by SPARQL query and the information given by the program logs confirms that LOD is loading data from database, mapping it to knowledge, inferring and serving it into the Web.

VI. CONCLUSION

In this paper we show the data repository of the European H2020 City4Age project, with the motivation of create a

smart environments in a context of the cities and improve the relationship between citizens and technologies. We explain the benefits of having a smart environment with some basic examples such us the improvement of the communication between citizens and cities or the detection of potential health risk like MCI or frailty.

We review the literature to present similar cases to our approach and prove that our work is a new way to gather and infer data in the context of the cities. The approach try to cover a new unknown area that need to be explored to give Linked Open Data new ways to improve the citizens life.

We present a new methodology to gather data from citizens and use Linked Open Data to share it using the concept of semantic Web. This new approach uses a rule based engine reasoner to infer new statements based in a set of rules that can be fit with the context of the citizens. The rules can be designed to create new RDF statements and try to create connections between elderly citizens and some common degenerative diseases. We explain the architecture of our approach, describing the most relevant technical tools used in the solution and giving reasons and motivations to use the described tools. We test all the architecture by using two different type of tests to 1) check if the architecture can gather real data from citizens of a big city; 2) check if our approach works as intended and the data is inferred and gathered to be shared by Open Data.

ACKNOWLEDGMENT

This work has been supported by the European Commission under the City4Age project grant agreement number: 689731.

REFERENCES

[1] U. Nations, "World urbanization prospects: The 2014 revision, highlights. department of economic and social affairs," *Population Division, United Nations*, 2014.

[2] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart cities in europe," *Journal of urban technology*, vol. 18, no. 2, pp. 65–82, 2011.

[3] J. M. Shapiro, "Smart cities: quality of life, productivity, and the growth effects of human capital," *The review of economics and statistics*, vol. 88, no. 2, pp. 324–335, 2006.

[4] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[5] A. Almeida and D. López-de Ipiña, "A distributed reasoning engine ecosystem for semantic context-management in smart environments," *Sensors*, vol. 12, no. 8, pp. 10 208–10 227, 2012.

[6] J. Vazquez, A. Almeida, I. Doamo, X. Laiseca, and P. Orduña, "Flexeo: an architecture for integrating wireless sensor networks into the internet of things," in *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*. Springer, 2009, pp. 219–228.

[7] A. Almeida, D. López-de Ipiña, U. Aguilera, I. Larizgoitia, X. Laiseca, P. Orduña, and A. Barbier, "An approach to dynamic knowledge extension and semantic reasoning in highly-mutable environments," in *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*. Springer, 2009, pp. 265–273.

[8] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "Owl 2 web ontology language primer," *W3C recommendation*, vol. 27, no. 1, p. 123, 2009.

[9] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *Semantic services, interoperability and web applications: emerging concepts*, pp. 205–227, 2009.

[10] N. Komninos, *Intelligent Cities: Innovation, Knowledge Systems, and Digital Spaces*. Spon Press, 2002. [Online]. Available: https://books.google.com.uy/books?id=psQq2PJp07gC

[11] L. Torres, L. Pradas, V. Pina, V. Martínez, S. Royo, U. de Zaragoza. Facultad de Ciencias Económicas y Empresariales, and C. O. de Comercio e Industria de Zaragoza, *E-government and the Transformation of Public Administrations in EU Countries: Beyond NPM Or Just a Second Wave of Reforms?*, ser. Documentos de trabajo. Universidad de Zaragoza, Facultad de Ciencias Económicas y Empresariales, 2005. [Online]. Available: https://books.google.com.uy/books?id=dKaqygAACAAJ

[12] N. Komninos, "Intelligent cities: towards interactive and global innovation environments," *International Journal of Innovation and Regional Development*, vol. 1, no. 4, pp. 337–355, 2009.

[13] S. Alawadhi, A. Aldama-Nalda, H. Chourabi, J. R. Gil-Garcia, S. Leung, S. Mellouli, T. Nam, T. A. Pardo, H. J. Scholl, and S. Walker, "Building understanding of smart city initiatives," in *International Conference on Electronic Government*. Springer, 2012, pp. 40–53.

[14] E. Theodoridis, G. Mylonas, and I. Chatzigiannakis, "Developing an iot smart city framework," in *Information, intelligence, systems and applications (iisa), 2013 fourth international conference on*. IEEE, 2013, pp. 1–6.

[15] A. Ojo, E. Curry, and F. A. Zeleti, "A tale of open data innovations in five smart cities," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on*. IEEE, 2015, pp. 2326–2335.

[16] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, and K. Moessner, "Enabling smart cities through a cognitive management framework for the internet of things," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 102–111, 2013.

[17] P. Barnaghi, W. Wang, L. Dong, and C. Wang, "A linked-data model for semantic sensor streams," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 468–475.

[18] S. Bischof, A. Karapantelakis, C.-S. Nechifor, A. P. Sheth, A. Mileo, and P. Barnaghi, "Semantic modelling of smart city data," 2014.

[19] R. Petrolo, V. Loscrì, and N. Mitton, "Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms," *Transactions on Emerging Telecommunications Technologies*, 2015.

[20] G. Nemirovski, A. Nolle, Á. Sicilia, I. Ballarini, and V. Corado, "Data integration driven ontology design, case study smart city," in *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*. ACM, 2013, p. 43.

[21] aitoralmeida/c4a_data_repository. [Online]. Available: https://github.com/aitoralmeida/c4a_data_repository

[22] L. Mainetti, L. Manco, L. Patrono, I. Sergi, and R. Vergallo, "Web of topics: An iot-aware model-driven designing approach," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 46–51.

[23] E. Serral, R. Mordinyi, O. Kovalenko, D. Winkler, and S. Biffl, "Evaluation of semantic data storages for integrating heterogenous disciplines in automation systems engineering," in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*. IEEE, 2013, pp. 6858–6865.

[24] Flask documentation. [Online]. Available: http://flask.pocoo.org/docs/0.12/

[25] PEP 333 – python web server gateway interface v1.0. [Online]. Available: https://www.python.org/dev/peps/pep-0333/

[26] uwsgi 2.0 documentation. [Online]. Available: https://uwsgi-docs.readthedocs.io/en/latest/

[27] Nginx documentation. [Online]. Available: https://nginx.org/en/docs/

[28] C. Bizer and A. Seaborne, "D2rq-treating non-rdf databases as virtual rdf graphs," in *Proceedings of the 3rd international semantic web conference (ISWC2004)*, vol. 2004. Springer, 2004.

[29] A. Almeida and D. López-de Ipiña, "Assessing ambiguity of context data in intelligent environments: Towards a more reliable context managing system," *Sensors*, vol. 12, no. 4, pp. 4934–4951, 2012.

[30] Apache jena - reasoners and rule engines: Jena inference support. [Online]. Available: https://jena.apache.org/documentation/inference/

[31] Apache JMeter - apache JMeter. [Online]. Available: http://jmeter.apache.org/