A Demonstration of 4D Digital Avatar Infrastructure for Access of Complete Patient Information

Project acronym: MyHealthAvatar

Deliverable No. 6.3
Data and RDF repository and evaluation report

Grant agreement no: 600929

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
C O	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	MyHealthAvatar
Project Full Name:	A Demonstration of 4D Digital Avatar Infrastructure for Access of Complete Patient Information
Deliverable No.:	D6.3
Document name:	Data and RDF repositories and evaluation report
Nature (R, P, D, O)[1]	R
Dissemination Level (PU, PP, RE, CO)[2]	PU
Version:	1
Actual Submission Date:	07/03/2016
Editor: Institution: E-Mail:	Zhikun Deng University of Bedfordshire zhikun.deng@beds.ac.uk

ABSTRACT:
This deliverable focuses the implementation and evaluation data and RDF repositories that the data access and management of the MyHealthAvatar project. The report initially introduce the implementation and technology for data and RDF repositories, then report the evaluation of the repositories in the MyHealthAvatar peoject.

KEYWORD LIST:
Data repository, RDF repository, Repository implementation, Evaluation, Cassandra.

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600929.

The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

MODIFICATION CONTROL			
Version	**Date**	**Status**	**Author**
0.1	15/01/2016	Draft	Zhikun Deng Hong Qing Yu
0.2	31/01/2016	Draft	Zhikun Deng Hong Qing Yu
0.3	07/02/2016	Draft	Zhikun Deng Hong Qing Yu
0.4	21/02/2016	Draft	Zhikun Deng Hong Qing Yu
1	29/02/2016	Final	Zhikun Deng

List of contributors

- Zhikun Deng (BED)
- Hong Qing Yu (BED)
- Youbing Zhao (BED)
- Enjie Liu (BED)
- Feng Dong (BED)

Table of Contents

Table of Figures

1 Executive Summary

MyHealthAvatar is an attempt at a proof of concept for the digital representation of patient health status. It is designed as a lifetime companion for individual citizens that will facilitate the collection of, and access to, long-term health-status information. This will be extremely valuable for clinical decisions and offer a promising approach to acquire population data to support clinical research, leading to strengthened multidisciplinary research excellence in supporting innovative medical care. The data and RDF repository is a crucial part of the system, and it is implemented utilise cutting edge technologies with maintainability, reliability and scalability in mind.

This document reports on the data and RDF repository and evaluation of the repositories. Further to D6.2 design for data and RDF repository, this documents includes the implementation details of the repository as well as the technical evaluations.

Data repository is mainly based on Apache Cassandra cluster; the selection of Cassandra is based on previously investigation in D3.1 User Requirements and D6.2 design document.

RDF repository implements on top of Virtuoso, which is a change from D6.2's choice of OWLIM, this change is due to OWLIM rename to GraphDB [1][2] and withdraw free edition during project development (although at end of 2015 GraphDB released a free version [3], which however come with limitations that the free version cannot handle more than two queries in parallel, which make OWLIM/GraphDB not suitable to this project), we research the RDF repository technologies again and Virtuoso is chosen instead.

2 Introduction

Data collection, access, management and share play a crucial role in MyHealthAvatar project, basic healthcare related data are important to both individuals who manage their own health and also very useful to clinicians and other healthcare workers in patient monitoring and providing suitable in-time care.

The implementation of repositories takes into account the scalability requirements of life logging data as well as reliability of the data storage in disaster events, e.g. hard drive failure. Since large amount of data is constantly coming lots of users' various data sources, it is fair to treat the data repository in the context of Big Data. Bearing in mind the number of users is increasing and the quantity and types of the information are emerging that will be stored on daily basis for the duration of a person's entire life, the repository implementation tackles this ever increasing data with the ability to scale horizontally.

2.1 Structure of this document

This document describes the implementation of data and RDF repositories, then reports the infrastructure for data and RDF repositories for MyHealthAvatar, finally the evaluation of repositions and evaluation results.

The structure of this document is organised as follows. Section 3 and 4 describe the reason of the way implementation and architecture of data repository and RDF repository, respectively. Section 5 describes the infrastructure of repositories for MyHealthAvatar. Following that, Section 6 discusses the implementation and testing of the data repository access/API, section 7 describes the technical evaluation strategy and result on data repository. Section 8 concludes the report and outlines the future work.

3 Implementation of Data Repositories

NoSQL database solution Cassandra is selected for data repository according to the project requirements and nature of the data. Please refer to deliverable 6.2 for details of the database selection process.

3.1 Consistency of Apache Cassandra Database

Apache Cassandra implementations put high performance, high availability and scalability as its priorities for large volume data storage and query. Rather than traditional RDBMS database's ACID properties [4], Cassandra databases seek eventual consistency and apply Brewer's CAP theorem [5].

Theoretically it is nearly impossible to fulfil all three requirements of the CAP-Theorem and suggests that only two of the three different aspects of scaling out are can be achieved fully at the same time.

- **Availability**: All of the servers will always return data they have (even if it's not the latest data or consistent across the system).
- **Partition Tolerance**: The system continues to operate as a whole even if individual servers fail or can't be reached
- **Consistency***:* All the servers in the system will have the same data so anyone using the system will get the latest data regardless of any updates happening.

The current NoSQL database implementation for repositories follow different combination of the C, A, P from the CAP theorem.

- **CA**: Single site cluster, in which all nodes are always in contact. When a partition is required, the systems will block.
- **CP**: Part of the data might not always be available, however all the rest of data is always consistent and accurate.
- **AP***:* System is still accessible under condition of portioning, but the returned data might not be consistent from different nodes.

In general Cassandra is typically classified as an API system, which means that partition tolerance and availability are considered to be more important aspect than the consistency factor. However, Cassandra can also be tuned with replication factor as well as consistency level to more or less meet the consistency (C) [6].

Cassandra's AP characteristic means it is a week consistent structure, which eventually reaches a consistent state during data replication. When the latest version of data initially reaches on node in the whole Cassandra cluster, the older versions of the data are still available on other nodes (which means access Cassandra might not return consistent result at this point of time), but eventually all nodes will get the latest version and reach consistency.

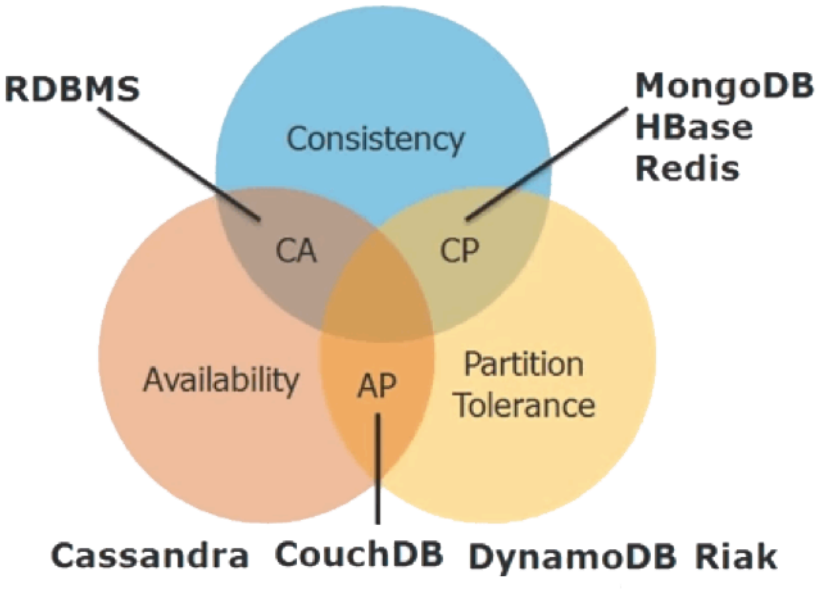


Figure 1 Cassandra and the CAP Theorem

To be more specific on the consistency, let:
R = Read replica count
W = Write replica count
N = Replication factor
Q = QUORUM (Q = N / 2 + 1)

Cassandra provides consistency when R + W > N, and in configuration of Cassandra, you have different level of consistency level:
ONE: means R or W is 1 (W = 1 or R = 1)
QUORUM: means R or W is ceiling (W = Q or R = Q)
ALL: means R or W is N (W = N or R = N)

So if we want to write with a consistent level of ONE and then get the same data from nodes when read, we will need to read Cassandra with consistent level ALL. Typical consistency configurations:

- W = 1, R = N
- W = N, R = 1
- W = Q, R = Q

In MyHealthAvatar configuration, the last situation is considered suitable, and implemented in the cluster environment as described in the architecture deliverable.

3.2 Apache Cassandra Architecture

This chapter mainly describes the Cassandra Architecture, which explains the implementation decisions of data repository of MyHealthAvatar.

3.2.1 Relation Model vs. Cassandra Databases

Cassandra is a distributed, highly scalable, eventually consistent, structured key-value stores use a hash table where a unique key is used to refer to a particular item of data. Cassandra provides a column family based data model, which is richer than typical key-value systems.

Cassandra data model is a schema-optional, column-oriented, which means that we do not need to model all of the columns required by the MyHealthAvatar project up front. Each row is not required to have exactly set of columns.

Table 1 provides a summary on RDBMS and Cassandra databases model, Cassandra keyspaces is an analogous to database, and column families is an analogous to the tables in relational database model.

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Table 1: RDBMS vs. Cassandra Model

3.2.2 Architecture of Cassandra

The table's meta-data information of RDBMS is normally stored and maintained in system tables; table creation and column name information are stored in system tables only. Analogue to this RDBMS, in Cassandra system key space is used to store the meta-data of other key spaces. The stored information includes cluster, local node information as well as operational data and hinted handoff information. It mainly consists of following:

- The nodes token – If node is supposed to take the value from 1 - 10. The node token is something like 1.
- The cluster name – It is not possible to bring nodes of two cluster names together, the cluster name defines to which the node belongs to.
- Key space and schema definitions to support dynamic loading – The metadata about different columns so Cassandra can support dynamic loading.
- Migration data – Key space configuration changes are normally stored in this category. E.g. if the replication factor is changed from 2-3 and 1-2, the information is stored under migration data
- The node is bootstrapped or not – When new node joins a cluster, the process of the new node get information about the cluster is called bootstrapping. Once a node is bootstrapped, it then starts copying the data from other nodes.

System key space cannot be modified or edited, and a system key space has two families:

- Schema column family: holds the schema definition and the other user key space data.
- Migration column family: records changes made to the key space.

3.2.3 CommitLog, Memtable and SSTable of Canssandra

This section focus on the internals of Cassandra storage implementations, to support Cassandra's fast write and durability requirements, the storage architecture of Cassandra is consiste of CommitLog, Memtable and SSTable. The CommitLog is a crash recovery mechanism and Cassandra writes to CommitLog first before writing to the Memtables.

There is a threshold of Memtable, when there is enough writes in the Memtables, the contents of the Memtable are flushed to disk files, which is called SSTable.

To indicate if data needs flushed or not within CommitLog, there is a internal bit flag for this purpose. Please note that once a Memtable is flushed to the SSTable, it becomes immutable which cannot be changed.

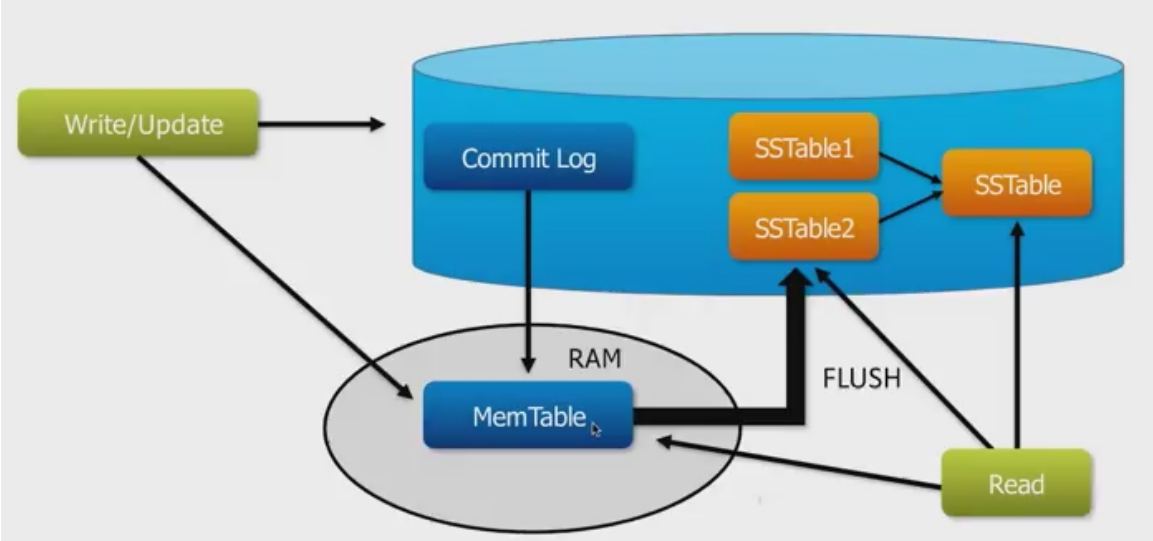


Figure 2 CommitLog Memtable and SSTable

3.2.4 Data Repository Architecture in MyHealthAvatar

As describe in D6.2 MyHealthAvatar overall data and RDF repositories design can be viewed as the figure below. As you can see that the Cassandra data repository sits in the middle, which not only stores incoming data and also provide data to other MyHealthAvatar module through the Restful APIs. The Restful API allows different modules to communicate in a distributed manner and allow different modules to use different programming languages.

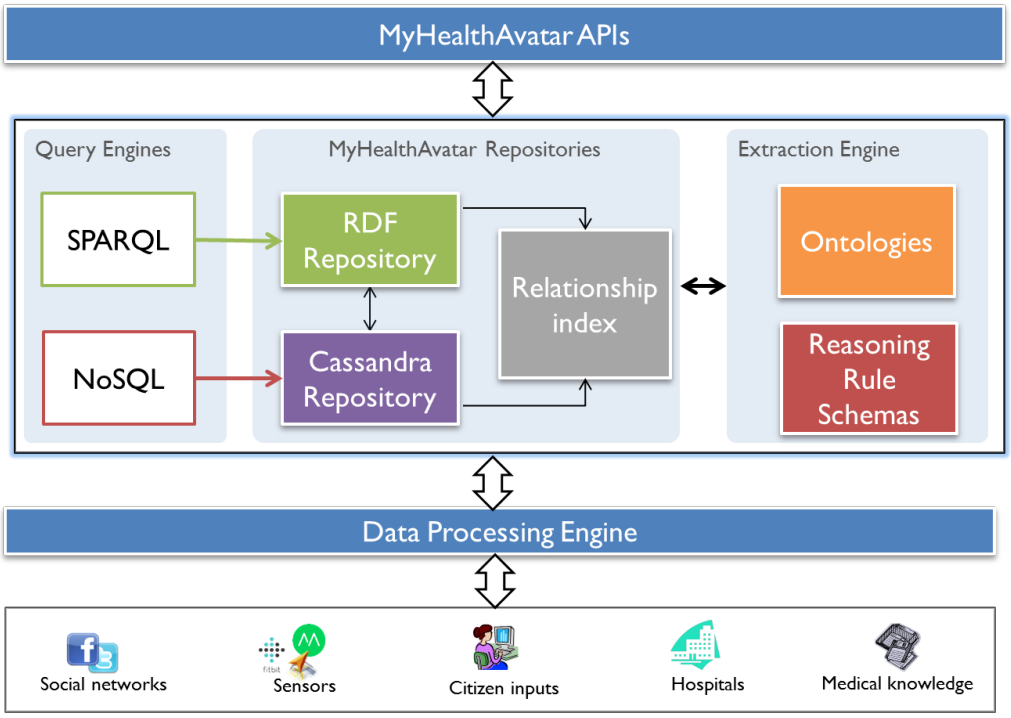


Figure 3 overall structures of MyHealthAvatar repositories

The data is coming from various data sources of MyHealthAvatar through the data repository interface into Cassandra data store. The data source to Cassandra looks like follows:

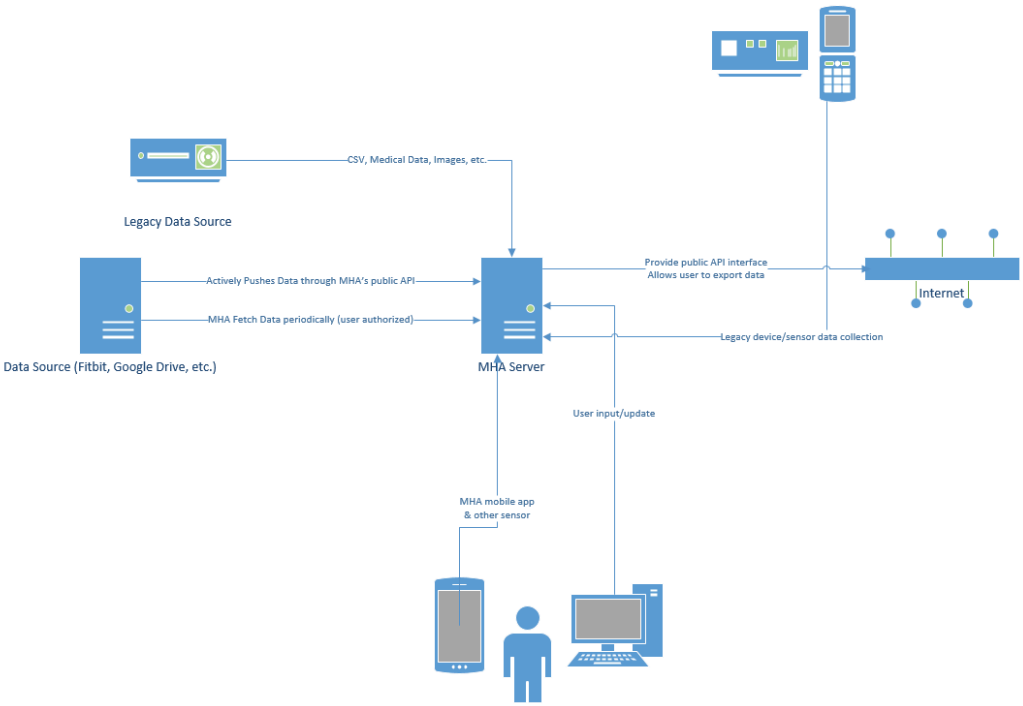


Figure 4 data flow to Cassandra in MyHealthAvatar

MyHealthaAvatar project has built a Cassandra cluster cross private and public cloud as a proof of concept of MyHealthAvatar data repository capability, which also servers the purpose of evaluation of Cassandra as the data store. The detailed infrastructure information can be found in Chapter 5 of this deliverable, and for most detailed information regarding the cloud service of MyHealthAvatar, please refer to the submitted architecture deliverable.

The main structure of the Cassandra deployment is depicted in following figure:

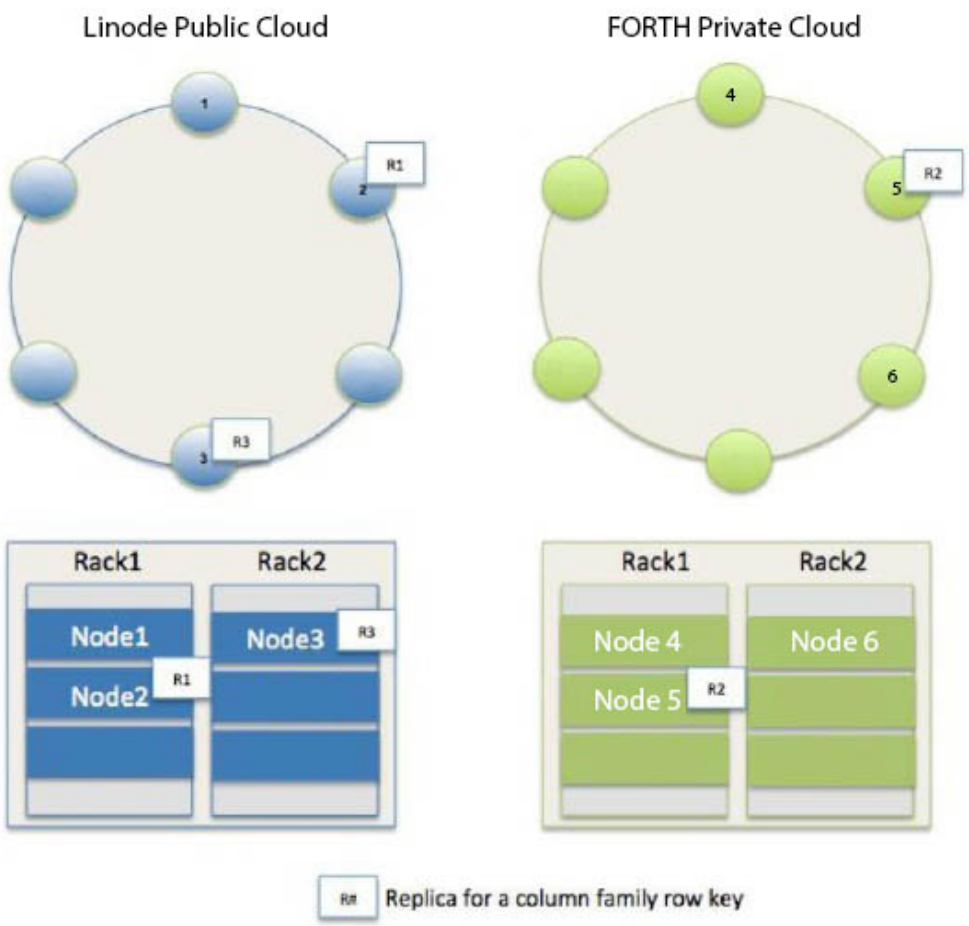


Figure 5 Cassandra cluster cross clouds

As you can see from above figure, the Cassandra nodes are distributed in both public and private cloud, while within respective cloud environment itself, nodes are deliberately distributed on different server racks. The configuration allows 3 replications of every piece of information come into the system, as we can see from above figure, R1, R2, R3 are on separate cloud (physical environment, network connection, etc.) and separate server rack (separate physical server and separate hard drive). All the deployment is to cope with the potential disaster situation, if power cut in one city, MyHealthAvatar's data repository should be still working as we have distributed the cluster in different geographical locations. In case of hardware failure of a server rack within one data centre, it should not affect the operational of the data repository since we have replications on different server racks.

4 RDF Data Collection and Integration

4.1 Data collection and integration

Collecting and meaningfully integrating of heterogeneous data resources is a longstanding problem in data management and engineering research area. In our research, we collect desired data from multiple data resources including mobile applications (e.g Moves), wearable sensors or digital measuring devices (Fitbit and Withings) and MHA platform. Each different data resource provides different and useful data information as Table III.A shows. The data collection process applies Web API technologies following OAuth security protocol. Whenever the user login to the MHA platform, the data from other devices can be synchronized into the system.

Data Collection	Sources	Data type
Steps	Fitbit/Moves/ MHA app	Count
Travelling & activity type	Fitbit/Moves/ MHA app	Minutes & transport type
Location	Moves/ MHA app	Coordination
Diet	MHA app	Calories & food category
Health Profile	MHA app – profile input	PHR like records
Weight	Withings scale	Grams
Body fat	Withings scale	Grams
Blood pressure	Withings BPM	mmHg
Slept hours	Fitbit/Withings	minutes
Awoken times	Fitbit/Withings	Count
Social activity	MHA app – calendar	Description

Figure 6 data resources

In Deliverable 6.2, we already discussed the advantages of using NoSQL databases for data integration. In MHA, the column-based NoSQL database (Cassandra) has been developed to mashup the heterogeneous data as whole. The Figure 7 shows the detail designed NoSQL database structure that bases on the column and key query data storing mechanism.

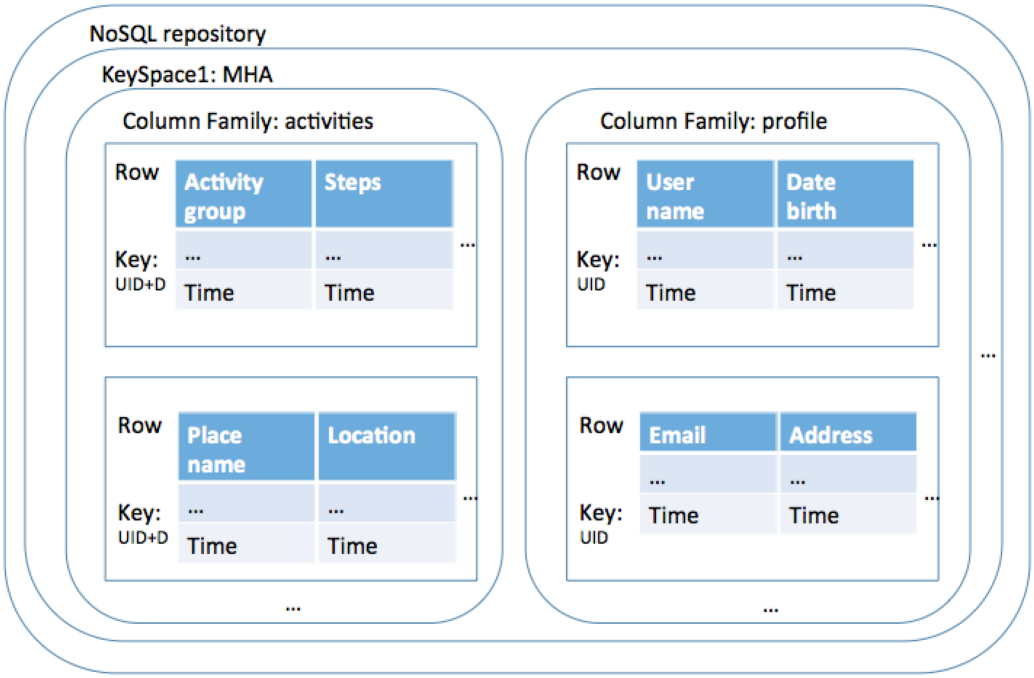


Figure 7 NoSQL data structure

Each Column family stores a group of rows that contains a set of individual columns in a specific data-structuring requirement. For example, one row in “activities column” groups all the data columns that stores activity type, step counting and duration data elements. The other row in the family can store the places information that the user has been travelled to or planned to visit. The “profile” column family completely focuses on managing user basic profile information such as name and contact.

4.2 Semantic ontology update

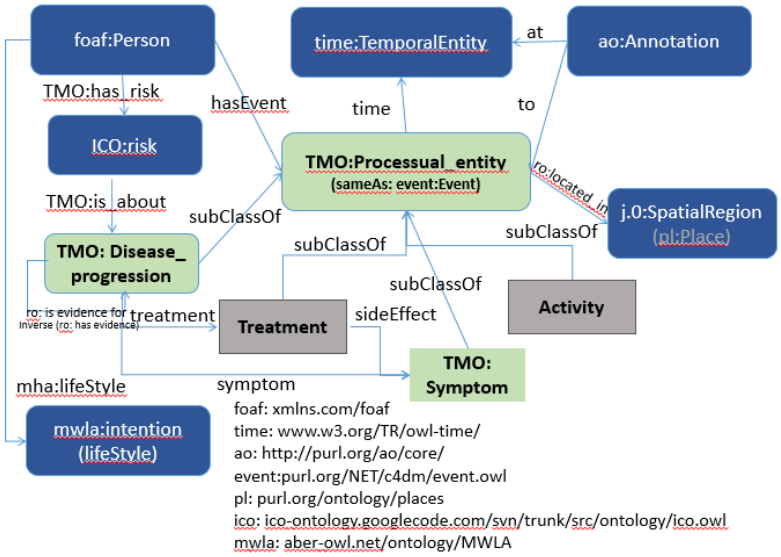


Figure 8 MyHealthAvatar Ontology

The core concepts of MHA H-event ontology includes 10 major terms as Figure xxx represents. The ontology extends TMO terminologies with some existing semantic concepts from well-known domain ontologies and our defined personal activity and treatment terms.

- **Event** defined as same as **TMO.Processual_entity** is a super concept to classify an interesting event that related to the health of an individual user. The event is the supper class of (discover a) **Symptom**, (taking a) **Treatment**, (diagnosed a) **TMO.Disease_progression** and (Having a) **significant activity**. Each event associates to a particular time point on user's time.TemporalEntity. In addition, Event is the central point of the whole ontology, which can be detected from data mining layer.
- **Person** is the concept to describe a MHA user using FOAF ontology. The FOAF ontology includes all possible aspects about a general profile of a person such as health history, gender and height. In our proposed framework, the semantic layer will nominalize user's name and address information that will be stored in lower level NoSQL database with more secured data management infrastructure and we will not discuss the security topic in this paper.
- **Significant activity** is a subclass of **Event** concept to identify the activity that is more significant to the user rather than includes all daily activities. In general, all the significant event should be related to understand user's health situation or life style. The activity type can be grouped by the exercise type such as "Running", "Driving", and "Shopping" but also can be categorized by the places and social activity type. Each significant activity should also record the time duration, places and possibly with distances, calories consumptions and steps.
- **Symptom** imported from TMO is a subclass of Event concept to present the unusual health related condition that are detected and concluded from the user's data. As same as all other events, the symptoms have to have a time stamp and places. Currently, the subclasses of Symptom include low/high blood pressure, unusual heart rate, unwell sleeping and significant weight/fat changes. Other unsenserable symptoms can also be added but have to rely on user's manually inputs.
- **Annotation** defined in AO (Annotation Ontology) is used as a semantic vocabulary link to an event. The annotations should use controlled vocabularies or semantic identifiers to define the meaning. The annotation can be automatically added through linked data annotation engine e.g. DBpedia spotlight or can be added by users via annotation tools from MHA platform.
- **Treatment** is a subclass of Event concept for recording the treatments that have been taken by the user from medical health organization or user-self. The treatment refers to any medical actions that have been done to the user such as taking dugs, operation and physical and mind therapies. In addition, the treatment requires identifying the exact time point on user's timeline.
- **Disease_progression** reused from TMO is a subclass of Event concept and presents the medical situations that were diagnosed in pass according the user's timeline or will be a potential risk for the user. The Health Condition, Treatment and Symptom concepts structure a triangle relation that could be a very valuable knowledge for individual user or a group of users.
- **Risk defined by ICO** is used as a concept to evaluate the possibility or progress levels to a particular health condition.

- **Lifestyle** is imported using intention concept in MWLA [] ontology that defined 25 lifestyle instances. Since MWLA is still a live EC project (CARRA []), the numbers of the lifestyle definitions can be enriched in the future.

4.3 RDF repository implementation: Virtuoso

A triple store is the data warehouse to store and query RDF data. A triple store should provide a mechanism and a set of APIs for persistent storage and access of RDF graphs. In order to decide and design the most suitable RDF repository solution, the comparing of different existing RDF repository technologies will be evaluated in this section.

According to the user requirements presented in Section of 5.1.4 and 5.1.5 of Deliverable 3.1, 4 important criteria have been identified:

(1) RDF, RDFS and OWL-based data reasoning and inference capabilities.

(2) Support SPARQL query

(3) Support large data set loading

(4) Query efficiency

In D6.2, we argued that OWLIM OWLIM-SE may be the ideal choice, but it can be obtained free of charge for evaluation purposes only. Therefore, the practice process requires to be divide into 2 steps. However, OWLIM product has been re-brand to GraphDB and fully commercialised in late 2015. Therefore, our original plan to use OWLIM is very cost on both time and financial terms. By further research, Virtuoso becomes the best repository that we can use to implement RDF triple store. The most recently comparing scores for GraphDB and Virtuoso are 0.08 and 2.82, which means the Virtuoso performs better than GraphDB. Virtuoso is a native triple store API provided by OpenLink Software. It is available in both open source and commercial licenses and provides command line loaders, a connection API, and support for SPARQL and web server to perform SPARQL queries and uploading of data over HTTP. Virtuoso is scalable to the region of 15.4 Billion triples. In addition to this, it provides bridges to be used with other RDF Data frameworks such as Jena and Sesame.

Since the Virtuoso can directly connect to the Jena framework, the semantic reasoning can be efficiently performed by combining Virtuoso and Jena APIs.

4.4 Repository implementation

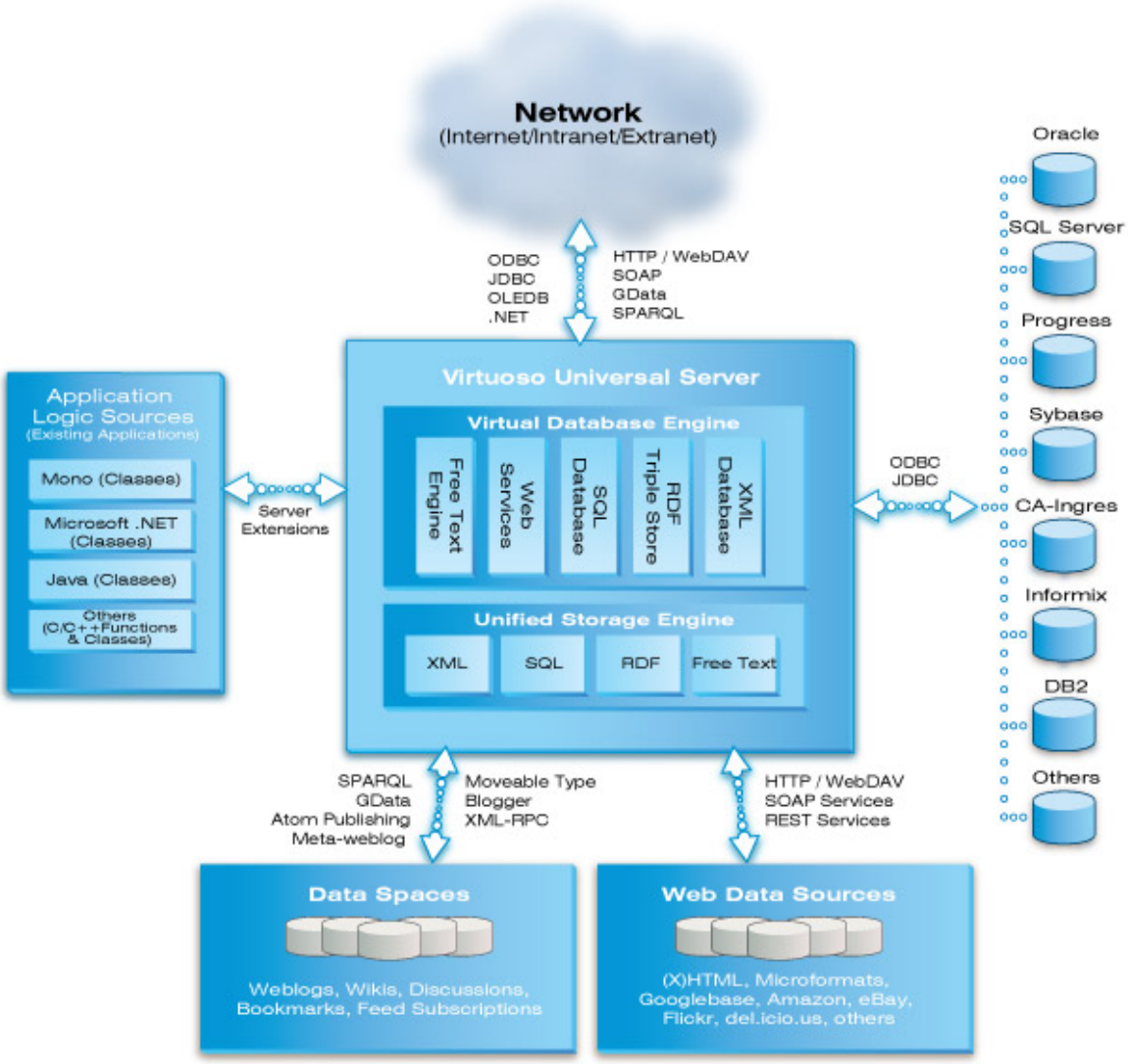


Figure 9 Virtuoso general architecture[3]

4.4.1 Installation Requirements

A typical installation will require a minimum of 400Mb of hard disk space to install the code, samples, documentation and sample database. The database will need additional space for data inserted, backups, logs and reports, web pages, etc.

The size of the database .db file will not reduce when data is removed. The spare space will however be reclaimed for later use.

[3] http://virtuoso.openlinksw.com/images/varch625.jpg

4.4.2 Operational Requirements

The Virtuoso database requires a minimum of 64Mb of system memory for each instance to operate in. Each connection will take between 70kb and 130kb of memory.

- ServerThreads
- ServerThreadSize
- MainThreadSize
- FutureThreadSize
- NumberOfBuffers

4.4.3 Operating System

We setup the Virtuoso server runs on Linux operation system which is a part of our cloud infrastructure.

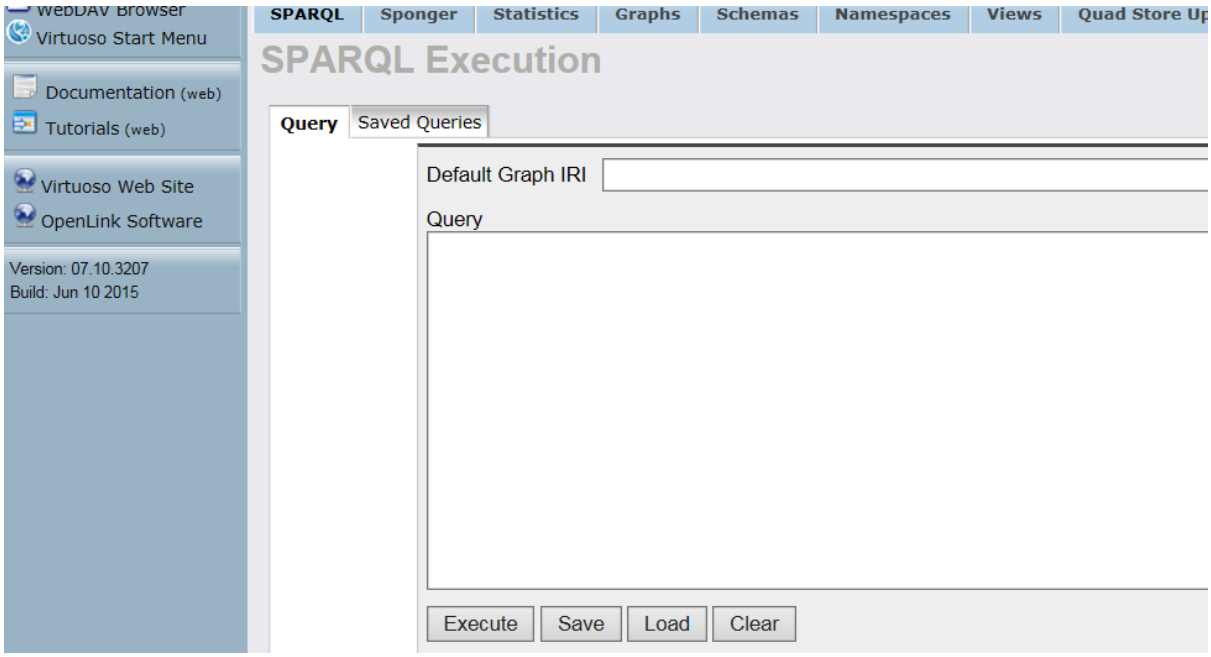


Figure 10 Virtuoso SPARQL Endpoint

5 Overall infrastructure of repositories for MyHealthAvatar

As previously describe in Chapter 3 of data repository, the repositories are deployed cross Linode public cloud infrastructure and FORTH private cloud infrastructure. In terms of hardware resources, the cloud infrastructure allows for maximum elasticity and flexibility by effectively adapting to the load of any given time. For availability and disaster resistance, MyHealthAvatar tends to utilise different server racks/hardware to host the data and RDF repository nodes. Please refer to following table for the main hardware of VMs MyHealthAvatar are using:

Type	Linode Public	Linode Public	FORTH Private	FORTH Private
Location	London	London	Crete	Crete
CPU	6 Core	4 Core	Xeon E5-2690	Xeon E7520
RAM	8G	4G	4G	4G
HARD DRIVE	192G SSD	96G SSD	100G	100G

Table 2 hardware summary of infrastructure

In terms of capacity of hosting provider, FORTH private cloud's current minimal specifications include: 300 GB of RAM, 9TB of storage and 16 cores Intel® Xeon® Processor E5-2690 and 4 cores Intel® Xeon® Processor E7520 (Dell PowerEdge R720 and SC 1425 Servers series). The software the OpenStack open source cloud computing software has been installed on the machines using the Linux Ubuntu 12.04 operating system. Linode as a public cloud provider, which is very capable of expand the VMs on demand, Linode virtual machines are mainly based on KVM technologies and Ubuntu 14.04 LTS 64bit operation system is the one MyHealthAvatar uses from Linode's choice of operation systems.

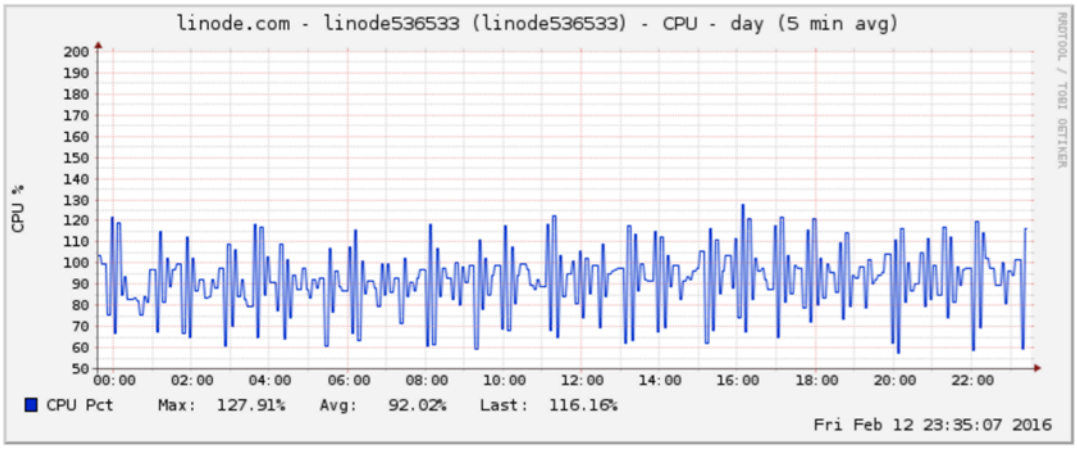


Figure 11 Linode CPU Usage Monitoring Graph

As you can see from above figure, that MyHealthAvatar are utilizing the capacity of the services properly, so that the VMs are not over kill for our purpose nor are too week for the demand, as you can see from above CPU usage graph.

Linode provides a web interface, which allows remote control of the VM's configuration through browser. The main panel looks like following graph, which provides normal operations in an intuitive manner. While the interface does not look very fancy compared to other provider (e.g. Microsoft Azure), it serves its purpose for MyHealthAvatar project well.

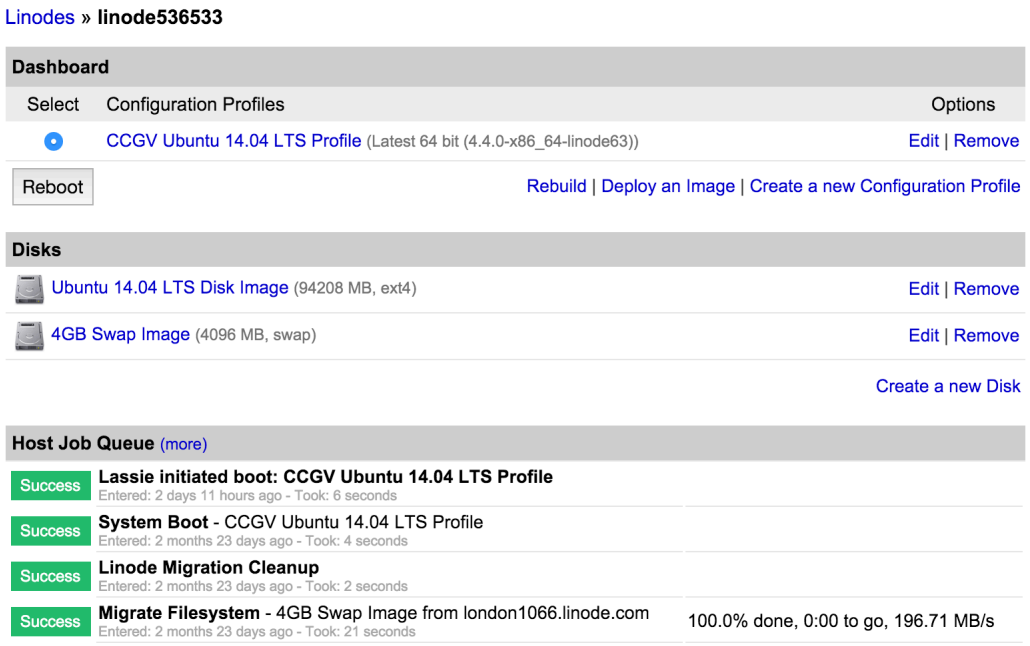


Figure 12 Linode Main Management Interface

While building the Cassandra cluster cross two regions, the connection legacy and speed are one of the concerns, the evaluation shows that fast connection within Europe continental make the cluster works very well. The speed test from MyHealthAvatar using speedtest.net service is:

mha@ccgv:~/mha/ccgv-social-java$ speedtest-cli
Retrieving speedtest.net configuration...
Retrieving speedtest.net server list...
Testing from Linode (178.79.142.72)...
Selecting best server based on latency...
Hosted by Vorboss Limited (London) [2.52 km]: 5.57 ms
Testing download speed..... Download: 630.94 Mbit/s
Testing upload speed..... Upload: 153.61 Mbit/s

With above network speed, which transfers data more than 15M bytes per second both ways, our infrastructure is sufficient for our cluster.

6 Repositories Implementation and Testing

Since most of MyHealthAvatar is implemented on top of Java Technology and Spring Framework, the data repository part is chosen to utilise compatible technology stack in order for better cooperation and easier integration.

Spring Data Cassandra offers a familiar interface as other Spring Data libraries, which makes it easier to get start with. The Spring Data Cassandra supports high level annotated POJOS for model the data while provide high performance data ingestion capacities. The main features of Spring Data Cassandra include:

- Support both synchronous and asynchronous data operations
- Support asynchronous call-back
- Support JavaConfig for all cluster and session capabilities
- Based on the latest DataStax Enterprise CQL Java Driver

6.1 Configuration and Implementation details

JavaConfig is used to configure Cassandra for the data repository access and API module, main configuration as follows:

```
@Configuration
@PropertySource(value = {"classpath:application.properties"})
@EnableCassandraRepositories(basePackages = { "uk.org.ccgv.repository" })
public class CassandraConfig extends AbstractCassandraConfiguration { ... }
```

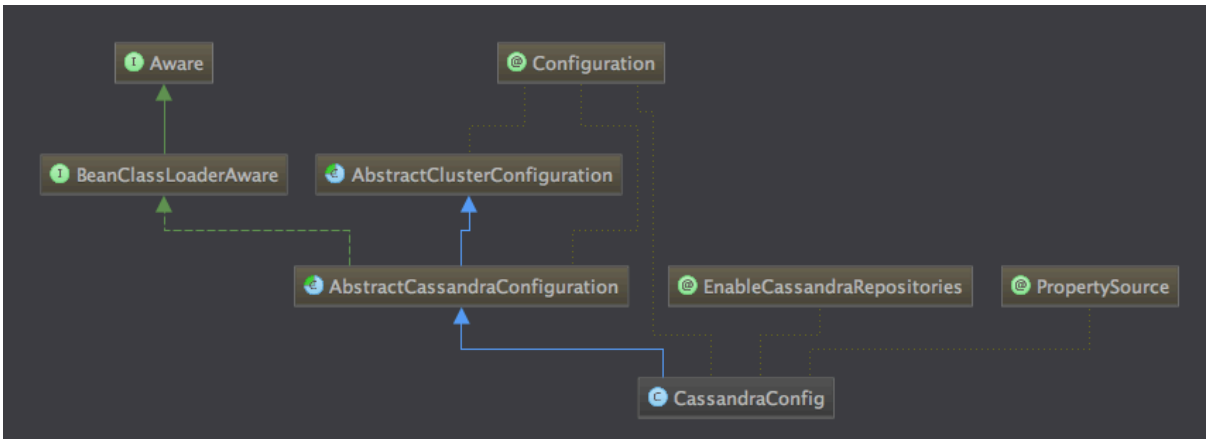


Figure 13 Cassandra Configuration Inheritance Hierarchies

6.2 User Profile Data Repository

The whole source code of the data repository implementation will be made available through open source license and in this report we will take user profile as an example to explain how the things work together in the context of MyHealthAvatar.

First of all, CQL query is used to create the main profile table in Cassandra:

```
 // Note that many columns definitions are omitted to be concise
CREATE TABLE mha.user_profile (
   user_name text,
   last_update timestamp,
   timestamp timestamp,
   description text,
   diabetes boolean,
   dob text,
   gender text,
   height decimal,
   surname text,
   title text,
   weight decimal,
   …
   PRIMARY KEY (user_name, last_update)
) WITH CLUSTERING ORDER BY (last_update DESC);
```

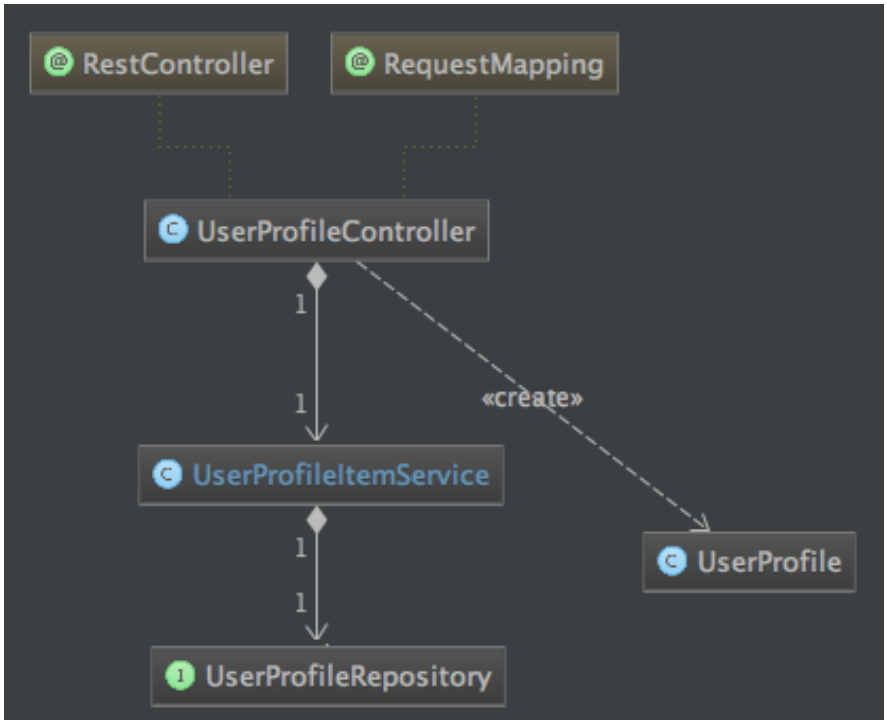


Figure 14 Main Data Repository Class Relationship

The above figure displays the relationship of the main class for data repository access, which is loss coupled from each other by utilize dependency injection techniques. Pure plain Java object (POJO) is preferred for the data repository implementation, one which is created to map between Cassandra table and Java object looks like follows:

```
@Table("user_profile")
public class UserProfile {
```

```
    @PrimaryKey
    private KeyUsernameTime update;
    ...
}
```

A repository class is created which utilize the POJO above and talks to the Cassandra:

```
public interface UserProfileRepository extends CrudRepository<UserProfile, KeyUsernameTime> {
    @Query("select * from user_profile where user_name=?0")
    Iterable<UserProfile> findAllByUsername(String username);

    @Query("select * from user_profile where user_name=?0 order by last_update desc limit 1")
    UserProfile findLatestByUsername(String username);
}
```

Object oriented programming language Java is designed to abstract to allow code sharing and reduce duplications; the project implementation uses latest best practice and design patterns to achieve high quality code. Following service layer class is a good example of our abstraction using generics:

```
public class UserProfileItemService<T extends UserProfileItem, S extends
UserProfileItemRepository<T> & CrudRepository<T, KeyUsernameTime>> {

    @Autowired
    protected S repository;

    public T saveForUser(T item, String username) {
        if (null != item) {
            if (null == item.getUpdate()) {
                KeyUsernameTime update = new KeyUsernameTime(username);
                item.setUpdate(update);
            }

            if (!username.equals(item.getUpdate().getUsername())) {
                item.getUpdate().setUsername(username);
            }

            item = repository.save(item);
        }

        return item;
    }
    ...
}
```

The above service class then get injected to controller which handles incoming GET, POST, DELETE, PUT requests:

```
@RestController
```

```
@RequestMapping("/profile")
public class UserProfileController { ... }
```

In order to make sure that the API services works as design and according to the user requirements analysis, unit tests and integration tests are written to help make sure that crucial parts of the system always work the way they supposed to. As you can see from following figure, that all test cases are run and make sure 100% success before artifact (war files in Java) is generated.

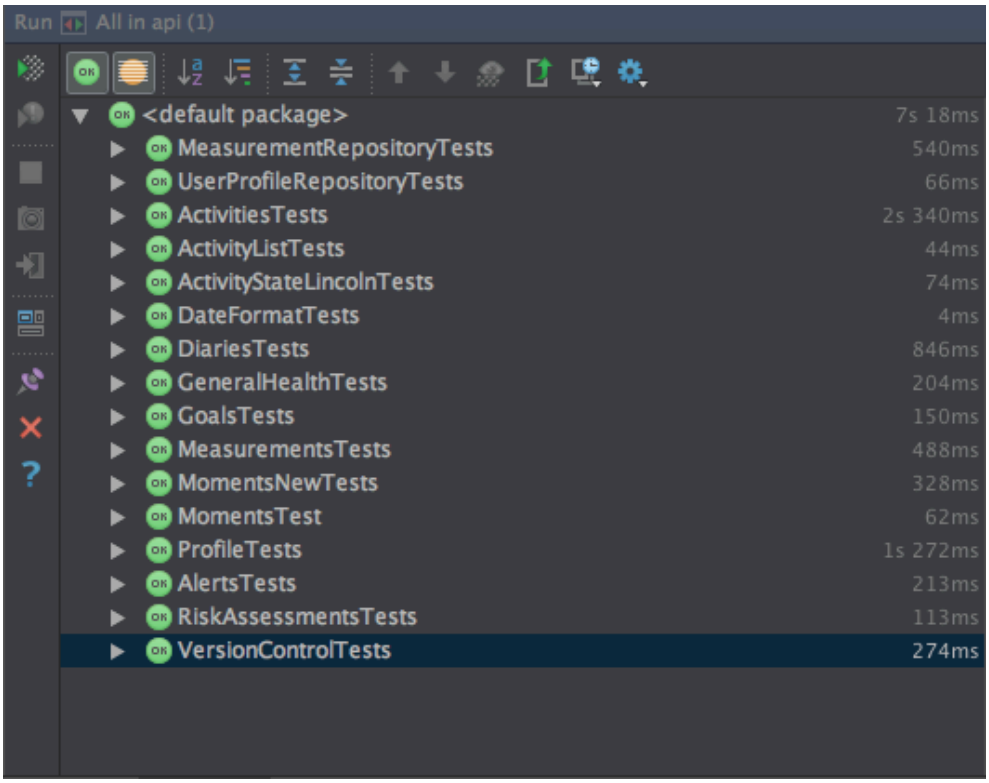
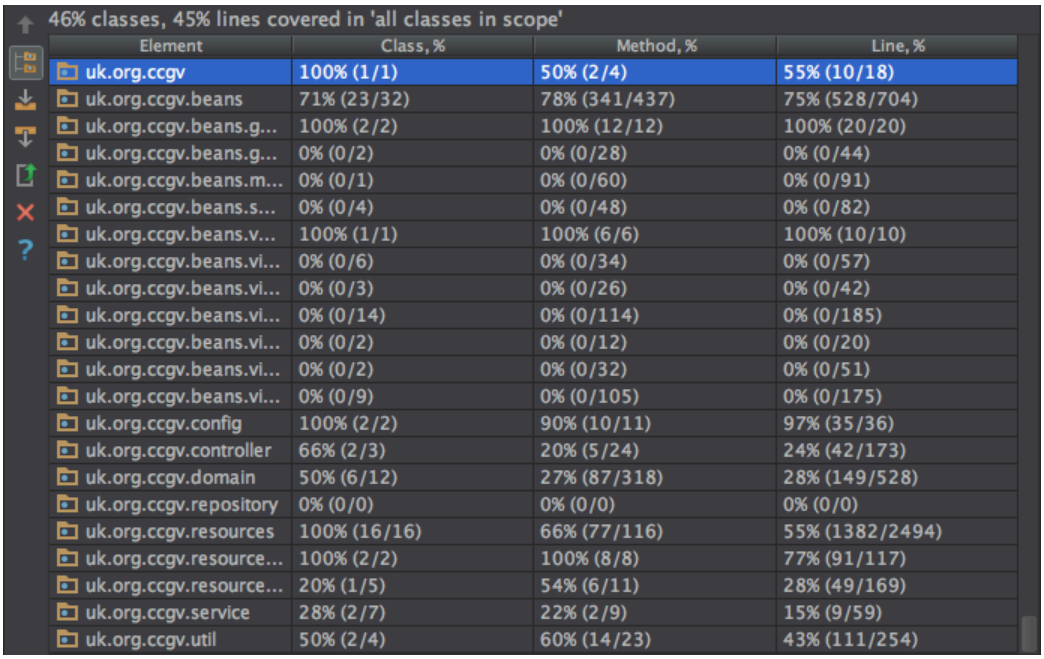


Figure 15 Unit Tests for Technique Evaluation

Software engineering best practice aims to have 100% code coverage from the testing cases, however due to the nature of the project, lots of effort is on the main functionality and features of the system. All core parts of the repositories are covered by unit tests, and the data repository project has code coverage of around 45% lines. If resources are to be allocated for further development, the project will aim to have a hundred percent code coverage for data repository.

In order to evaluate the data repository's functionality and performance, most of the test cases are writing as integration test manner rather than pure unit tests. Which means that at beginning of each test class set up stage, a connection to the testing full Cassandra instance is made, the sequential tests are run against live instance of Cassandra database rather than mocked Java services. This allows the best compatibility between development, evaluation platform and public facing production platform. The only drawback is that this process slows down the build speed of the data repository project; however, the benefit outweighs this little more time spend.



46% classes, 45% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
uk.org.ccgv	100% (1/1)	50% (2/4)	55% (10/18)
uk.org.ccgv.beans	71% (23/32)	78% (341/437)	75% (528/704)
uk.org.ccgv.beans.g...	100% (2/2)	100% (12/12)	100% (20/20)
uk.org.ccgv.beans.g...	0% (0/2)	0% (0/28)	0% (0/44)
uk.org.ccgv.beans.m...	0% (0/1)	0% (0/60)	0% (0/91)
uk.org.ccgv.beans.s...	0% (0/4)	0% (0/48)	0% (0/82)
uk.org.ccgv.beans.v...	100% (1/1)	100% (6/6)	100% (10/10)
uk.org.ccgv.beans.vi...	0% (0/6)	0% (0/34)	0% (0/57)
uk.org.ccgv.beans.vi...	0% (0/3)	0% (0/26)	0% (0/42)
uk.org.ccgv.beans.vi...	0% (0/14)	0% (0/114)	0% (0/185)
uk.org.ccgv.beans.vi...	0% (0/2)	0% (0/12)	0% (0/20)
uk.org.ccgv.beans.vi...	0% (0/2)	0% (0/32)	0% (0/51)
uk.org.ccgv.beans.vi...	0% (0/9)	0% (0/105)	0% (0/175)
uk.org.ccgv.config	100% (2/2)	90% (10/11)	97% (35/36)
uk.org.ccgv.controller	66% (2/3)	20% (5/24)	24% (42/173)
uk.org.ccgv.domain	50% (6/12)	27% (87/318)	28% (149/528)
uk.org.ccgv.repository	0% (0/0)	0% (0/0)	0% (0/0)
uk.org.ccgv.resources	100% (16/16)	66% (77/116)	55% (1382/2494)
uk.org.ccgv.resource...	100% (2/2)	100% (8/8)	77% (91/117)
uk.org.ccgv.resource...	20% (1/5)	54% (6/11)	28% (49/169)
uk.org.ccgv.service	28% (2/7)	22% (2/9)	15% (9/59)
uk.org.ccgv.util	50% (2/4)	60% (14/23)	43% (111/254)

Figure 16 Code Coverage by Tests

Most of the class for data repository implementation and evaluation is under uk.org.ccgv.* namespace, as you can see from above running result, the code coverage from these tests are around 45% at time of writing this documents.

7 Evaluation Strategy and Report

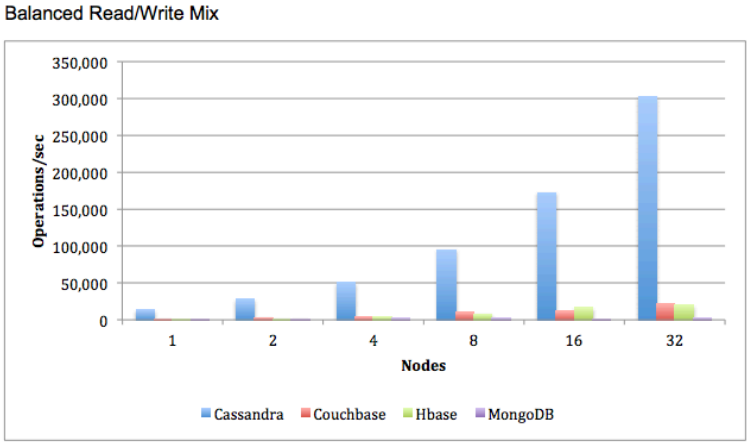
The main focus of the evaluation is to find out if the design and implementation of the data and RDF repository has satisfied the user requirements of MyHealthAvatar projects. There are user evaluations and feedbacks of using the system (data and RDF repositories) in a separate deliverable (D9.4). This chapter's main focus is on technique evaluation, which verifies the repositories by functionality questionnaires, integration tests feedback (automatically and manually), and also benchmarks of the repositories.

7.1 Evaluation Strategy and Tools

The evaluation of the repositories can be roughly looked from mainly four aspects capability, functionality, stability and usability. Four different methods are used to evaluate the repositories from a top down approach, from overview into details.

- Third party testing results for parallel compare with other NoSQL
- Unit tests to cover functionalities
- Cassandra-stress tool to load test and benchmarking
- Questionnaire for simple user evaluations

There are plenty of third parties tests result [7] compare Cassandra with other NoSQL databases which is designed for similar purpose. All tests we believe are bias in one way or another, and those tests are more or less tailored for one specific database. However due to limited capability and resources allocated, it is not quite possible for the project to run all the tests against major NoSQL solutions. Rather third party with similar user scenario to MyHealthAvatar's prospect future, and details of the configuration looks more or less fair to all the databases are chosen as a reference of the capability of MyHealthAvatar repository.



Nodes	Cassandra	Couchbase	HBase	MongoDB
1	13,929.58	1,554.14	527.47	1,278.81
2	28,078.26	2,985.28	1,503.09	1,441.32
4	51,111.84	3,755.28	4,175.8	1,801.06
8	95,005.27	10,138.80	7,725.94	2,195.92
16	172,668.48	11,761.31	16,381.78	1,230.96
32	302,181.72	21,375.02	20,177.71	2,335.14

Figure 17 Cassandra advantage in balanced Read/Write [7]

We briefly described about Unit tests in previous chapter of this documents, and following is a specific example of how the test is set up for evaluate the repositories functionality.

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = ApiApplication.class)
@WebIntegrationTest({"server.port=0", "server.context-path=/api"})
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class AlertsTests { ... }
```

As above code piece shows that annotation based meta-programming is the key for the Spring Junit tests, it is written as web integration test, which allows real data go through the whole data repository process logic and then operate against real Cassandra instances.

```
@Value("${local.server.port}")
int port;

@Before
public void setUp() throws IOException, URISyntaxException {
   URI jsonFileURI = this.getClass().getResource("/alerts-2015-11-02.json").toURI();
   alertsJson = new String(
         Files.readAllBytes(
               Paths.get(jsonFileURI)
         )
   );
   RestAssured.port = port;
}
```

Each test class has a set up phase, which has the data repositories temporary port passed in as variable, at the set up stage, testing JSON data is read in as Java String to be reused across different test cases. Test case themselves uses fluent Java API, which make the code readable as if they are the description of the functionality of MyHealthAvatar to be tested.

```
@Test
public void canT30GetLatestAlerts() {
   expect()
      .body(
            "update_time", hasSize(1)
      ).
   when()
      .get(Configure.contextPath + "/user/general_alert/latest?username=" + username)
   .then()
      .content(Matchers.not(Matchers.containsString("[{")))   // return result is not array, to update
      .content(Matchers.containsString("update_time"))
```

```
    .statusCode(HttpStatus.SC_OK);
}
```

Tools are used to stress test and benchmark the repositories, for data repository Cassandra, a tool named cassandra-stress from official Cassandra source repository is used. Cassandra-stress is a tool for benchmarking and load testing a Cassandra cluster, it supports testing arbitrary CQL tables and queries to allow benchmark of MyHealthAvatar data model in several operation types:

- write-only, read-only and mixed workloads of standard data
- user configured workloads, running custom queries on custom schemas
- write-only and read-only workloads for counter columns

Example of running the tool against one node looks like following:

tools/bin/cassandra-stress write n=10000000 -node 192.168.1. 1 # 10M inserts to given host
tools/bin/cassandra-stress read n=10000000 -node 192.168.1.1 -o read # 10M reads

Technique peers have filled in the questionnaire regarding the usability of the repository within MyHealthAvatar environment as well as directly through API interface (as third party applications), the results are listed in 7.3 below.

7.2 Evaluation Environment and Configuration

The evaluations are done in the environment of virtual machines mainly in Linode public cloud environment, the hardware is described in Chapter 5 of this document. Ubuntu Linux Server 14.04 LTS 64bit is the operation system for repositories, Tomcat 7/8 over Java 8 64bit is the runtime environment for repository interface. Operation system has all latest security patches applied, and Cassandra we are evaluation is version 2.1.5 with CQL spec 3.2.0 and native protocol v3. And Virtuoso Open Source Edition v7.10.3207 (see figure 10 for details) is the RDF repository installed. Details of the environment are queried as follows:

```
mha@ccgv:~$ uname –a
Linux ccgv 4.4.0-x86_64-linode63 #2 SMP Tue Jan 19 12:43:53 EST 2016 x86_64 x86_64 x86_64
GNU/Linux

mha@ccgv:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.5 | CQL spec 3.2.0 | Native protocol v3]

mha@ccgv:~$ java –version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)

mha@ccgv:~$ /usr/share/tomcat7/bin/version.sh
Server version: Apache Tomcat/7.0.52 (Ubuntu)
Server built:   Jun 19 2015 08:54:46
```

Server number: 7.0.52.0
OS Name: Linux
OS Version: 4.4.0-x86_64-linode63
Architecture: amd64
JVM Version: 1.8.0_45-b14
JVM Vendor: Oracle Corporation

7.3 Evaluation Results

Integration functionality tests all successfully run (please refer to figure 15) each build of the repository code; this is set as default. This evaluation is running very frequently as part of repositories continuously integration process, it guarantees that all functionality is working as expected by integration test cases. Not only the existing test cases are required to run and success each build, MyHealthAvatar also adopted the software engineering techniques of Test Driven Development (TDD). For new features, test cases are written first as functionalities description, according to the way that new functionalities are supposed to work. Since the new features are not yet implemented, the newly written test cases will initially fail to run. Then new features are implemented according to the test cases, and when it fulfils all test cases and make all test cases successfully run, at this point that all desired functionalities are implemented properly.

Functionality and usability is a short questionnaire for peer technicians/programmers who used the MyHealthAvatar repository and/or the repositories directly through Restful API interfaces. Due to the nature of the small targeting group, there is not much statistical analyses for the results, rather useful and import findings are listed as follows:

Feature	Expected Functionality	Actual
Store Activity Data	User can store activity data import from Move, Fitibt, Withings, etc. through Synchronise button of MyHealthAvatar interface	The storage works find, no error found during the synchronise process
Retrieve Activity Data	Stored activity data should be able to view on MyHealthAvatar interface, including diary and life tracker panel.	The activity can be displayed on the interface correctly, the rendering sometimes can be slow
Set goals data through API	Third party application developer should be able to post data to data repository, including goals API	Goals stored successfully when post data with OAuth 2 token
Get goals data through API	Third party application developer should be able to retrieve data from data repository, including goals API	Retrieve goals data stored in repository successfully with user authorisation
Repositories API documents	All features of public repositories API are documented and available to developers	Documents are available to read online, however some of endpoints description is out of date and need update

Table 3 repositories functionality evaluation results

As mentioned in previous chapter, that due to resource limitations, it is not feasible to cross benchmark all NoSQL solutions professionally and fairly. Instead third party organisations' testing result is adopted as reference of how the repositories performs in comparison to other similar solutions on markets.

Cassandra stress tests are run against the evaluation platform which is single node Cassandra in Linode London, this is due to legal requirements according to data protection regulations. During evaluation phase, user data need to be gather and store within UK. The tests are running use following command:

Write: *cassandra-stress write duration=15min -mode native cql3 -rate threads=700 -node 192.168.1.1*
Read: *cassandra-stress mixed 'ratio(read=1)' duration=15min -pop 'dist=gauss(1..10000000,5000000,500000)' -mode native cql3 -rate threads=700 -node 192.168.1.1*
Mixed: *cassandra-stress mixed 'ratio(read=1,write=1)' duration=15min -pop 'dist=gauss(1..10000000,5000000,500000)' -mode native cql3 -rate threads=700 -node 192.168.1.1*

And the results are as follows:

Workload	Average CQL tps	Min CQL tps	Max CQL tps
write	201,428	183,698	219,823
read	76,698	61,511	89,556
mixed	87,265	72,433	91,484

Table 4 Cassandra benchmark results

Open Source JMX for Enterprise Computing (mx4j) [8] plugin is used together with the StorageProxy MBean for above test results. As you can see from above testing result diagram, on the evaluation node, the data repository would support current user base (hundreds) without any problem.

8 Conclusion

This deliverable focused on the implementation and evaluation of MyHealthAvatar data and RDF repositories. The repositories are already implemented and supporting the various MyHealthAvatar evaluations activities for past months, the detailed results of repositories within the system could be found in relevant deliverables. Cassandra and Virtuoso are evaluated in this deliverable under the context of MyHealthAvatar. From the results gathered, the repositories design and implementation meet the MyHealthAvatar user requirements, and the scalability is suitable for the platform user base growth.

The flexible and hybrid approach of repositories in MyHealthAvatar, which combine the pros of NoSQL database Cassandra as well as benefits of RDF store Virtuoso. The best of both worlds' ability support the heterogeneous data of MyHealthAvatar well and enables future developments and researches on this firm foundation of platform.

9 References

[1] “Ontotext GraphDB FAQ” - https://confluence.ontotext.com/display/GraphDB6/GraphDB+FAQ

[2] “Sones GraphDB” - https://en.wikipedia.org/wiki/Sones_GraphDB

[3] “ GraphDB ver. 6.6 Release Features New GraphDB Free Edition” - http://ontotext.com/company/news/graphdb-ver-6-6-released-introducing-new-graphdb-free-edition/

[4] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery," ACM Computing Surveys (CSUR), vol. 15, no. 4, pp. 287-317, 1983.

[5] S. Gilbert and N. Lynch, “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," ACM SIGACT News, vol. 33, no. 2, pp. 51-59, 2002.

[6] “Cassandra Wiki” - http://wiki.apache.org/cassandra/

[7] “Benchmarking Top NoSQL Databases” - http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf

[8] “Open Source JMX for Enterprise Computing” - http://mx4j.sourceforge.net/

Appendix 1 – Abbreviations and acronyms

ACID	Atomicity, Consistency, Isolation, Durability
AO	Annotation Ontology
APIs	Application Interfaces
ARQ	A SPAQRL Processor for Jena
BMI	Body Mass Index
BSON	Binary JavaScript Object Notation
CAP	Consistency, Availability, Partition Tolerance
CFS	Cassandra File System
CHI	Consolidated Health Informatics
CT	Computed Tomography
CTO	Clinical Trial Ontology
DICOM	Digital Imaging and Communications in Medicine
DTO	Disease Treatment Ontology
DO	Disease Ontology
ECG	Electrocardiography
EEG	Electroencephalography
EHR	Electronic Health Record
EKG	Electrocardiography
EO	Event Ontology
FOAF	Friends of a Friend
GO	Gene Ontology
GridFS	Grid File System

HDFS	Hadoop Distributed File System
HTTP	Hyper Text Translation Protocol
iTQL	Interactive Tucana Query Language
J2EE	Java 2 Platform Enterprise Edition
JDBC	Java Database Connector
JSON	JavaScript Object Notation
KB	Knowledge Base
KBN	Knowledge Base Network
MeSH	Medical Subject Heading
MRI	Magnetic Resonance Imaging
MRSI	Magnetic Resonance Spectroscopy Imaging
MVCC	Multiversion Concurrency Control
NCI	The National Cancer Institute
NCIt	The National Cancer Institute Thesaurus
NoSQL	No only SQL
NSAID	Nonsteroidal Anti-inflammatory
NSCLC	Non-Small Cell Lung Cancer
OAuth	Open Authentication
OBI	The Ontology for Biomedical Investigations
OCRe	Ontology of Clinical Research
OWL	The Web Ontology Language
OWL-DL	The Web Ontology Language – Description Logic
PATO	Phenotypic Quality Ontology

PET	Positron emission tomography
PRO	PRotein Ontology
PROEVO	Protein Evolutionary Classes
RESP	Redis Serialization Protocol
REST	Representation State Transfer
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SCLC	Small Cell Lung Cancer
SNOMED CT	SNOMED Clinical Terms
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SPECT	Single Photon Emission Computed Tomography
SQL	Structured Query Language
TMO	Translational Medicine Ontology
VPH	Virtual Physiological Human
XML	Extensible Markup Language