DEVELOPING AN ONTOLOGY EXPLORATION APPLICATION FOR MOBILE DEVICES

By

ALEXANDROS DAMIANAKIS

BSc, Technological Education Institute of Central Greece, 2014

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF ENGINEERING

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2019

Approved by:

Professor Nikos Papadakis
Professor Haridimos Kondylakis

Abstract

In this study, we developed an innovating Android application in order to display data from OWL ontologies. Using some tested technologies for Android development the application became stable and usable. Moreover, we created some new algorithms in order to decode, store and handle OWL ontologies at mobile devices.

The first chapter is the introduction of our study. We summarize the structure of the project and we mention the general approach of our concept.

At the second chapter exists a demonstration of the related work. All articles, tutorials, and applications that we found and they are related to our topic. We learned what technologies have been used and helped us to find the important issues of that kind of applications.

The next chapter includes the main concept of the application. Is presented the system architecture, is analyzed the basic functions of application and is mentioned the used software tools. It is an analytic demonstration of all application operations.

The fourth chapter has the usability evaluation of the application. It includes some experiments of user interaction with the application and some performance results that help us to gain some useful conclusions.d

Finally, the last chapter is about the total conclusion of our project and some suggestions for future work at related topics.

Content table

Chapter 1

Introduction

1.1 Introduction

In computer science, ontology is a representation of the categories, properties, and relations between data.

Ontologies are one of the building blocks of Semantic Technologies. They provide the necessary structure to link one piece of information to other pieces of information on the Web of Linked Data. Their ability to describe relationships make them the bases for modeling high-quality, linked data.

In recent years, there has been an uptake of expressing ontologies using ontology languages such as the Web Ontology Language (OWL).

Another technology that we will study is the Android software. Android is one of the most fashionable, popular and easy-to-use software platforms. It has very interesting software components and tools.

In nowadays, Android is the most used operating system in mobile devices for the reason that is independent of the hardware device structure. Moreover, Android devices bring ever more impressive software to our hands. In addition to upgraded processors and RAM, mobiles are becoming more "smart" with the addition of sensors that perceive almost everything.

1.2 Scope of the project

In this project is studying OWL ontologies in order to display their data at mobile devices. Has been Analyzing a mechanism for decoding OWL data, storing and search them. By using tools from the Android platform we developing a mobile application in order to handle OWL ontologies.

The general purpose of this project is to develop a mobile application which user can make queries to OWL ontologies and gain the proper information’s. In order to be a useful and practical application, the queries that the user will be done will be triggered by a simple click or typing a few letters.

That kind of application is useful if a user needs to have fast access at some data by using only a smartphone. Of course, the format of these data should be an OWL ontology. Same applications operate but only for the desktop environment, for mobile devices, all applications with this concept are in the experimental stage.

1.3 Design the application

The major benefit of mobile devices is that they are portable, so the user can use them from everywhere. However, they have an important defect too: is that the processing power is lower than desktop computers.

Considering that, in our application, we used some tested technologies at mobile devices in order to make the app stable and fast.

However, this was not enough for developing that kind of application. For that reason, should be included some innovating techniques.

In order to achieve our target, we studied some principles of the Semantic Web. We constructed some innovating algorithms to fit at mobile devices processing power. Finally, we designed the system architecture to have a decent speed performance.

1.4 The Ontology Explorer application

The name of our application is Ontology Explorer. By using this application user can explore the data of four OWL ontologies through a user-friendly UI. The operations that Ontology Explorer is doing are:

- Getting the ontology data from a remote server.
- Decoding and storing the data at a local database.
- Depending on the user interaction makes at the background the proper queries.
- Displaying the results at the user interface.

To sum up, this project combines a solution of decoding, storing and handling OWL ontologies that could provide new knowledge at the academic community with some interesting developing techniques in order to make the final application practically useful.

Chapter 2

Related work

2.1 Introduction

This chapter includes all the study that we have done before deciding the way that we developed our mobile application. We searched solutions and applications that have been made by other teams in order to get ideas, compare results and finally make a decision of what is the ideal method to implement our project.

Firstly we did a search in the academic field, especially we were read some papers in order to learn some basics about our topic and explore new methods that have solved problems. Secondly, we studied some tutorials on developing similar applications to get a more practical aspect. Finally, we discovered smart-phone and desktop applications associated with our topic. All these information's will be presented in this chapter.

2.2 Academic work

Below we list some academic work that has done on the basis of exploring a semantic ontology using a mobile device. All papers which have been searched about android devices for the reason that we have a bigger device variety to explore the results.

2.2.1 Android goes Semantic: DL Reasoners on Smartphones.

This paper shows that Android devices could be able to use "semantic reasoners" through some experiments. Semantic reasoners are software which is able to infer logical consequences from a set of asserted facts or axioms. [1][2]

Is used semantic APIs on Android devices like OWL API and Androjena in order to manage the OWL ontologies. The reasoners that have been used is JFact,CB,HermiT,Pallet . Moreover, they tried to load another 4 reasoners on Android without success because the android virtual machine (Dalvic) has unsupported classes. Dalvic is a discontinued process virtual machine in Google's Android operation system that executes applications written for Android. [3]

For experiments is Used 5 ontologies (Pizza,Wine,DBPedia,GO,NCI) that tested the classification performance of each reasoned on two android devices and on a PC.The experiment showed that Android devices could be able to use most of the semantic reasoners. The results (how time is the need for execution) are showing below:

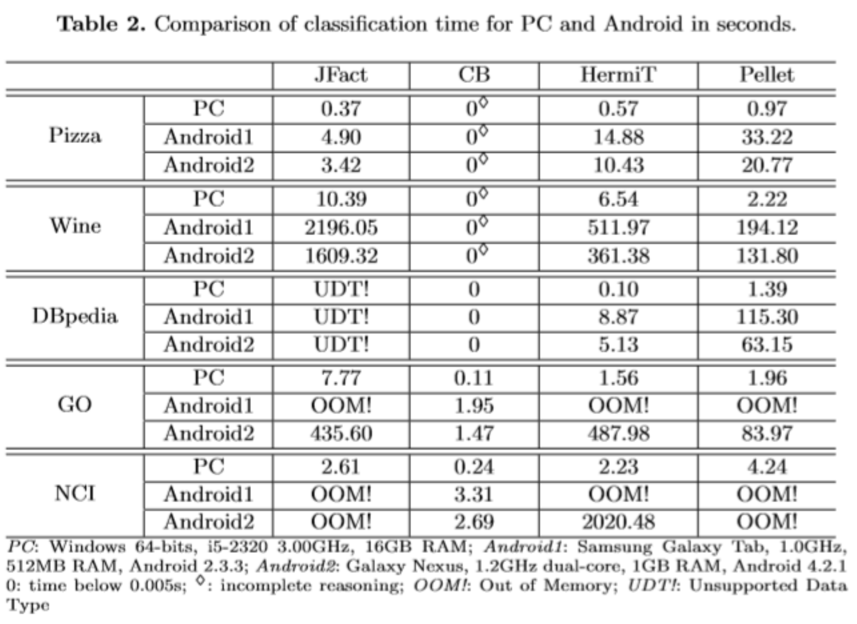


Table 2. Comparison of classification time for PC and Android in seconds.

		JFact	CB	HermiT	Pellet
Pizza	PC	0.37	$0^{\diamond}$	0.57	0.97
	Android1	4.90	$0^{\diamond}$	14.88	33.22
	Android2	3.42	$0^{\diamond}$	10.43	20.77
Wine	PC	10.39	$0^{\diamond}$	6.54	2.22
	Android1	2196.05	$0^{\diamond}$	511.97	194.12
	Android2	1609.32	$0^{\diamond}$	361.38	131.80
DBpedia	PC	UDT!	0	0.10	1.39
	Android1	UDT!	0	8.87	115.30
	Android2	UDT!	0	5.13	63.15
GO	PC	7.77	0.11	1.56	1.96
	Android1	OOM!	1.95	OOM!	OOM!
	Android2	435.60	1.47	487.98	83.97
NCI	PC	2.61	0.24	2.23	4.24
	Android1	OOM!	3.31	OOM!	OOM!
	Android2	OOM!	2.69	2020.48	OOM!

PC: Windows 64-bits, i5-2320 3.00GHz, 16GB RAM; *Android1*: Samsung Galaxy Tab, 1.0GHz, 512MB RAM, Android 2.3.3; *Android2*: Galaxy Nexus, 1.2GHz dual-core, 1GB RAM, Android 4.2.1
0: time below 0.005s; $^{\diamond}$: incomplete reasoning; *OOM!*: Out of Memory; *UDT!*: Unsupported Data Type

We consider that for each result needs a lot of time, the faster execution at the android device is 1.47 seconds and the longer execution is 20196.05 seconds. This kind of solution is not ideal in our application for the reason that each search of the ontology will need a lot of time to return some results. Perhaps this solution could work but it will not have the best performance in our application.

2.2.2 Android went Semantic: Time for Evaluation.

This paper has many similarities with the previous project that we demonstrated. So, is investigating the use of semantic reasoners on mobile devices. The differences from the previous paper are the new reasoners.[4]

In particular, is analyzing the recent versions of already considered OWL 2 DL reasoners and some new reasoners specific for the OWL 2 EL profile.

Moreover, become a classification of ontologies which they have been implementing to experiment. They are three categories of ontologies (small, medium,large) the classification depended on the size of each ontology.

The experiment became with a smartphone, a tablet, and a PC. As a result, was that reasoners are much faster in the PC. Furthermore, on Android devices during the classification happen frequently time outs and out of memory errors (in the OWL 2 DL profile and in larger ontologies, are most usual).

The solution of reasoners as we study from previous works does not fit with the functionality of the application that we will develop. The process of classification will be needed a long time, furthermore exists more issues about the stability of reasoners classification like time outs and out of memory errors.

2.2.3 DBpedia Mobile: A Location-Enabled Linked Data Browser.

This work is using a DBpedia dataset in order to develop a mobile application. The application is getting the current GPS position of a mobile device and shows a map which indicates nearby locations. The showing data (map and locations) are from DBpedia dataset which contains information about almost 300,000 locations.[5]

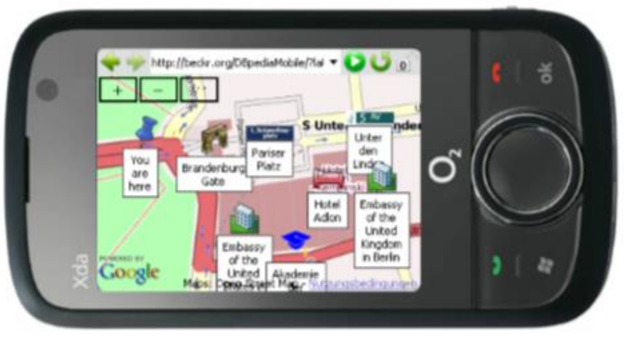


Figure 1: DBpedia Mobile's map view of resources in the user's proximity.

The application was developed by using JavaScript; is a client-server app so all background functions are processing at a remote Virtuoso server. Virtuoso server is making the proper SPARQL queries to DBpedia in order to get the data from the specific location. For the communication of client (mobile) and server (virtuoso) is responsible for a Java Servlet that generates XHTML views for given URIs resource.

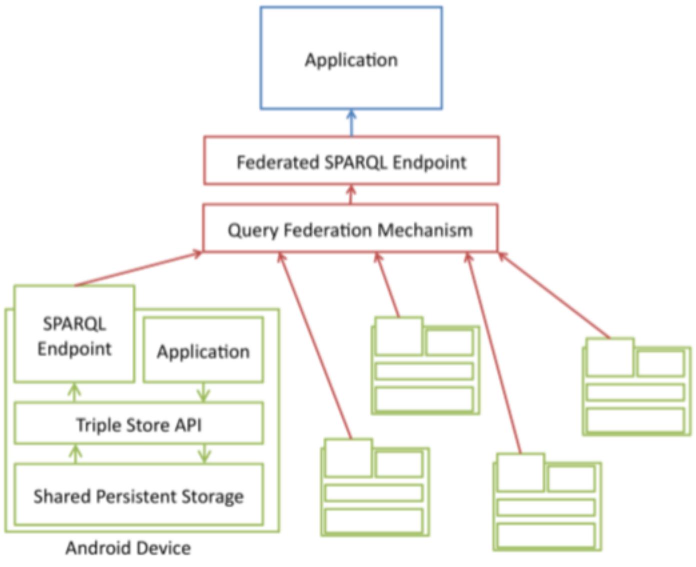
For using this application is a necessary network connection, moreover at this paper has no experiments about the speed of response. This client-server architecture perhaps is not the optimal solution to have a fast response from an OWL ontology.

2.2.4 Building SPARQL-Emabled Applications with Android Devices.

This paper describes a very interesting idea about a collaborative network of data sources. An android mobile application can store sensor data to the shared repository on the device in order to can be made accessible.[6]

Using an extended version of Seame library provides a persistent RDF store to the SD card of the device. In these RDFs will be stored the sensor data. Moreover is included a Web interface which is giving access to a SPARQL protocol.

With Web application server 'Jetty' has been provided a Web server and a servlet environment. This Web application implements a SPARQL endpoint. Finally, a federation of SPARQL query has been implemented in order to handle multiple SPARQL queries from deferent devices.


Application
Federated SPARQL Endpoint
Query Federation Mechanism
SPARQL Endpoint
Application
Triple Store API
Shared Persistent Storage
Android Device

In order to explain the above architecture, they developed a simple mobile application. The application can take pictures and using the extension of the Semantic Sensor Network ontology is storing at shared persistent Storage. With each picture is storing the following information's:

- The path of the picture at the device.
- The location of the device (using the GPS sensor).
- The time of taking it.
- The identifier of the device.

This application is tested with several different devices.

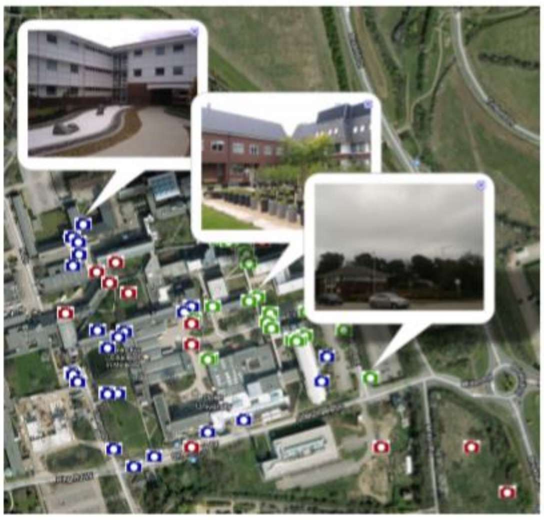


Fig. 3. Application mapping pictures from Android phones with SPARQL endpoints.

The basic concept of this project is very clever; the local storing of data is a very useful technique. However, the remote SPARQL endpoint will be a long time process.

For each time that the application makes a search is needed a remote SPARQL response from Web Application.

2.2.5 A linked data framework for Android

This paper presents an API that provides data access in RDF files. Through this solution, the applications will have a generic layer of data. Based on this idea is developing a framework which exposing application data as linked data. [7]

To achieve this target are using a little complex architecture using the following components:

- Activities -"is a crucial component of an Android app, the way activities are launched and put together is a fundamental part of the platform's application model". [8]
- Intents -"An intent is an abstract description of an operation to be performed".[9]
- Services -"is an application component that can perform long-running operations in the background, and it doesn't provide a user interface.[10]
- Broadcast receivers -"Base class for code that receives and handles broadcast intents send by Context.sendBroadcast(Intent)". [11]
- Content providers –"can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps".[12]

Using ContentProviders have been designed as an embedded API that offers access to RDF content.

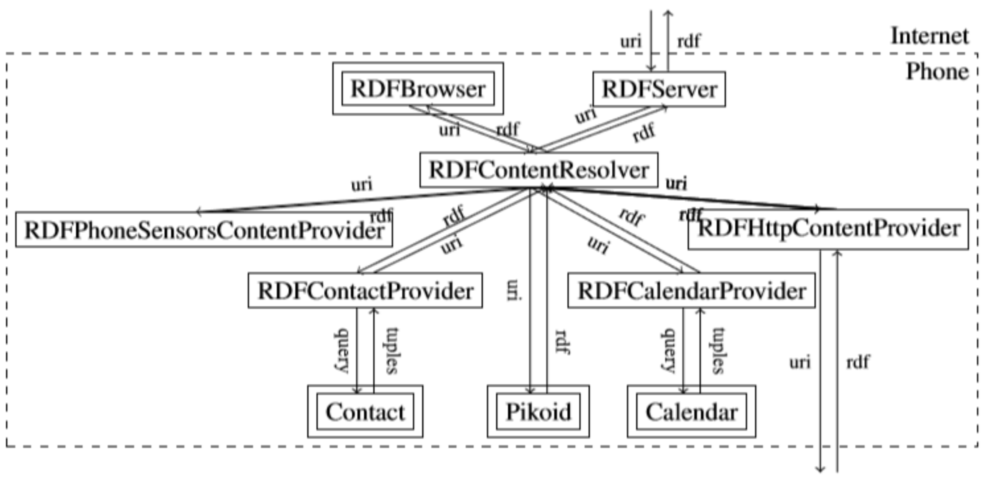


Fig. 1. The architecture components and the communication between them. Components with double square have a relevant graphic user interface.

The process of this architecture in the base is that Content providers have been scattering different features (HTTP requests, RDF schemas, Uri SPARQL endpoints, etc.). The data from a Content provider to another are sending by using Intents of Broadcast receivers components.

The RDF Content Provider using some saved URIs is getting the response of RDF data. After a successful receiving, a background service is getting the response (RDFContentResolver) in order to decide to which provider must redirect the query.

Another special feature that implements this framework is RDFServer. Is an HTTP server that takes incoming URIs and returns RDF to a remote RDF server. As a result, the remote RDF server exposes the stored RDF data to the outside world.

In order to prove the concept of this framework, this paper demonstrates some applications.

- An application that wrapped and manage the address book and the agenda as RDF ontology.
- An application that users could annotate pictures and answer some questions about the picture. All these operations have been done using native linked SPARQL queries to the local RDF file.
- An RDF Browser which can give to it a URI and then displays the RDF result. RDF Browser is a linked data client for stored RDF ontologies.

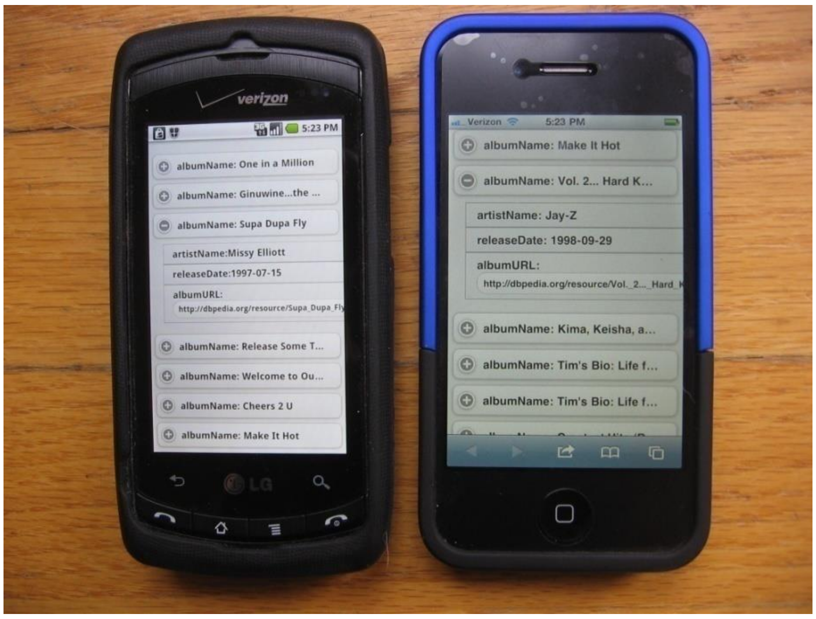
To sum up all these information's, this project is a very interesting work with many clever techniques and technologies. However, this solution has a complex architecture and an unknown value of the speed response. Considering the many queries and requests at embedded framework (from one Content Provider to another) perhaps is not so fast process. In our application, we definitely need a more flexible solution.

2.3 Online tutorials

In this section, we present some tutorials that have studied in order to build associated mobile applications. They helped us to get some ideas for development. Furthermore, we tried some of these technologies in order to end up at our own conclusions.

2.3.1 Displaying SPARQL results on a mobile phone

This tutorial describes a method in order to show SPARQL results on mobile. Using the jQuery mobile[13] (JavaScript library) to get SPARQL query results from a remote server. For rendering the results at mobile phone used an XSLT style sheet that can take the SPARQL Query Results XML Format. For the queries used a DBpedia dataset.[14]



albumName: One in a Million
albumName: Ginuwine...the ...
albumName: Supa Dupa Fly
artistName:Missy Elliott
releaseDate:1997-07-15
albumURL:
http://dbpedia.org/resource/Supa_Dupa_Fl
albumName: Release Some T...
albumName: Welcome to Ou...
albumName: Cheers 2 U
albumName: Make It Hot

albumName: Make It Hot
albumName: Vol. 2... Hard K...
artistName: Jay-Z
releaseDate: 1998-09-29
albumURL:
http://dbpedia.org/resource/Vol._2..._Hard_K
albumName: Kima, Keisha, a...
albumName: Tim's Bio: Life f...
albumName: Tim's Bio: Life f...

Is a very interesting approach to display SPARQL results on mobile phone and seems to be developed a nice User Interface. However, considering the technologies that have been used this application is a web mobile app. That means is necessary internet connection in order to work this application.

Moreover, we prefer to develop a native mobile app, so we will use different technologies (not JavaScript or XSLT UI). The native apps are faster and more efficient as they work in tandem with the mobile device. Also, they are assured of quality, as users can access them only via app stores online. [15]

2.3.2 Using ARQoid for Android-based SPARQL Query Execution

This tutorial presents two libraries to handle SPARQL queries against remote SPARQL endpoints. The libraries are Androjena's [16] and ARQoid [17]. [18]

The tutorial provides instructions to set up the libraries and has some examples of how to use the included classes. Furthermore, has a demo project which was available for download.

At first, seems this tutorial was solved a lot of our problems so we followed the instructions an implemented to our application the suggested libraries. But in the end, was not so simple.

The feature of these libraries is to make SPARQL queries from mobile to a remote SPARQL endpoint and getting the proper response. However, this never worked for our project. We tried some different queries at different endpoints with no response.

Finally, we noticed that the last update of these libraries was on the 5th of December 2010, so is not a new work. Furthermore in ARQoid page is informing the user with that kind of issues with the following message:" ARQoid is still under development and has been tested only with very simple queries, so don't blame us if it doesn't work and use source code and binaries AT YOUR OWN RISK!" [17].

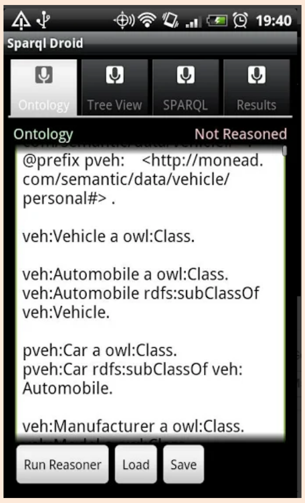
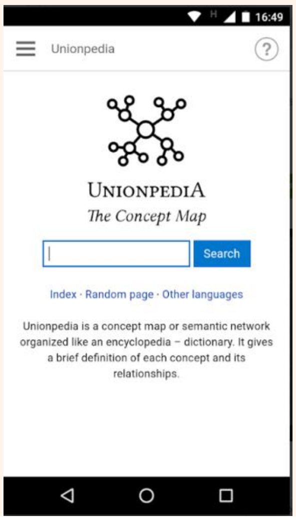
So these technologies are not helpful for our project.

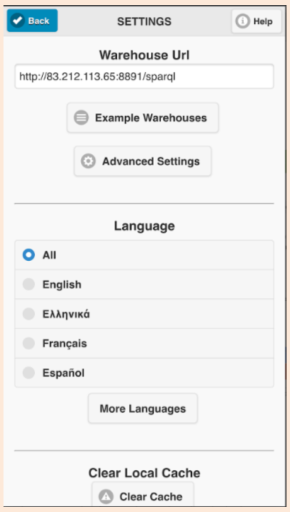
2.4 Related applications

After the study of academic projects and some practical tutorials, it was a vital thing to find applications that are working in real time. So we made research about applications which have been released in our research topic and we present them.

2.4.1 Android applications

In this chapter we present related android applications for our topic. There was not find easily this kind of applications. As you will see in this topic exists very few applications at Google Play. However, we found and we present them at the follow matrix.

Name of app	Description	Image
Sparql Droid	This application includes a few basic semantic technology features. The user provides ontology and then run SPARQL queries. In addition, a simple (and greatly constrained) tree-view of the reasoned model is provided. Is an early release which serves as a starting point and Proof-of-concept.[19]	 19:40 Sparql Droid Ontology Tree View SPARQL Results Run Reasoner Load Save Ontology Not Reasoned @prefix pveh: <http://monead.com/semantic/data/vehicle/personal#> . veh:Vehicle a owl:Class. veh:Automobile a owl:Class. veh:Automobile rdfs:subClassOf veh:Vehicle. pveh:Car a owl:Class. pveh:Car rdfs:subClassOf veh:Automobile. veh:Manufacturer a owl:Class.
Unionpedia	Unionpedia is a concept map for semantic network organized like an encyclopedic – dictionary. It gives a brief definition of each concept and its relationships. The android application is a network client that gets data from the web. It only works with an internet connection.[20]	 16:49 Unionpedia UNIONPEDIA The Concept Map Search Index · Random page · Other languages Unionpedia is a concept map or semantic network organized like an encyclopedia – dictionary. It gives a brief definition of each concept and its relationships.

PascoLink	Is a client-side SPARQL endpoint browser which supports a caching mechanism. The application connects with a remote ontology and using SPARQL queries a user can explore the ontology. The remote execution for each query is a very long time process and this process needs every time when a user searches something at ontology.[21]	

SETTINGS
Warehouse Url: http://83.212.113.65:8891/sparql
Language: [✓ All] English Ελληνικά Français Español

2.4.2 Desktop applications

We searched for some desktop applications of editing Semantic web ontology. The variety of applications in this field is bigger than the previous one. So, we chose the most popular and we present them. One of these applications we will use to do some test and evaluation of our application.

Application	Description
Protégé	Is a popular open-source ontology editor and framework for building intelligent systems.[22]
NeOn Toolkit	An open source multi-platform ontology engineering environment, which provides comprehensive support for the ontology engineering life-cycle.[23]
SWOOP	A small and simple ontology editor from the University of Maryland.[24]
TopBraid Composer	Is a paid Semantic Web editor which combines the world's leading semantic web modeling capabilities.[25]
Vitro	Is an integrated ontology editor and semantic web application.[26]
Knoodl	Contains tools for creating, managing, analyzing and visualizing RDF/OWL descriptions.[27]
Anzo for Excel	Is a semantic middleware platform and set of tools to build solutions leveraging the power of the W3C web standards.[28]
OWLGrEd	A simple graphical ontology editor for OWL.[29]
Fluent Editor	A comprehensive tool for editing and manipulating complex ontologies that use Controlled Natural Language.[30]
Semantic Turkey	Is an RDF service platform for Knowledge Management.[31]
VocBench	A web-based, multilingual, collaborative development platform for managing OWL ontologies, SKOS-XL Thesauri, Ontolex-lemon lexicons, and generic RDF datasets.[32]

2.5 Conclusion of related work

After studying related work, we pointed on two basic issues that are should be solved in order to develop a stable and useful application. We list these issues and we demonstrate the solution that we suggest and follow in order to develop our application.

2.5.1 The long time of operations.

At the related projects many operations are executing successfully but they need a long time. The reason for a long time processes is the using of SPARQL queries at mobile devices. Considering that Android devices have shorter memory and computing power compared with desktop PCs.

In our project, we solve this issue by using familiar technologies for android development. We use the SQLite database in order to store the ontology data to the device. Moreover, all the search operations in our application will be done by using SQLite queries. The process of SQLite queries is faster at mobile devices than SPARQL queries.

2.5.2 The necessity of an internet connection.

Most of the applications that we studied need an internet connection in order to execute their functions. The SPARQL functions are executing in a remote application server. Then the server sends the response at mobile application. This client-server architecture needs an internet connection for all operations.

In our application, we synchronize all data of the ontology from a remote server but only once. At the first time when the user visits the ontology is triggering an HTTP request which targeted at the remote server. With a successful response, all ontology data stored locally. After this operation, the ontology exists at the mobile device and the internet connection is not a necessity.

Chapter 3

System Architecture

3.1 Introduction

The application which we developed can provide a user-friendly interface to explore an OWL ontology. In this chapter, we present the technologies and the techniques included in order to achieve our target. We developed an application based on Semantic Web fundamental technologies by using android software development techniques.

3.2 Type of application and tools for developing

One of the things that we had to decide at the beginning of this project is what type of mobile app we will develop. There are three types of mobile applications:

1. Web apps
   Are hosted on web browsers and the users interact with the app through a web view. This type of app is easy to build and easy to maintain. However, are much slower than other application types (for example native apps), also much less interactive and intuitive.
2. Native apps
   Are built for specific platforms, are fast, responsive, offers intuitive user input and output and they don't require an internet connection. Overall offer a better user experience but are more difficult to the development.
3. Hybrid apps
   They are a combination of native and web apps. They consist of two parts: 1. Back-end code 2. A native shell that loads the code using a web view. Hybrid apps are a clever solution; however, they are slower than native apps and are not customizable to individual platforms. [33]

Considering all these we decide that the type of application which fit our purpose is the native mobile app. We need an application to execute fast the operations and the option to build a customizing user interface.

The second question that had to answer is in witch operating system will run our application Android, IOS or something else? (Windows,KaiOs, etc.). [34]

We decide to build an application that could use for most people. The first conclusion that we have reached is that Android applications could run in a bigger variety of devices than IOS. IOS operation system could run only in Apple devices; however, the Android operating system could run at 24.000 different devices. [35]

Moreover, we considered how many users have Android devices, how many have IOS devices and how many have other operating systems. We found the following statistics where they apply for May 2019:

- Android: 75,27%
- IOS: 22,74%
- Other operating systems: 1,99% [36]

Considering the above statistics is clear that most of the users are using Android devices. So we decided to develop a Native Android application.

They exist many tools in order to build a native Android application (Eclipse, ADB, AVD Manager, etc.). [37]

We decided to use the Android Studio because is the official integrated development environment, is free to download and is supported not only by Google but also by a large and active community of Android developers.

We downloaded it from the official site. The installation did not need some specific settings. However, in the official site exists installation guidelines in order to do this operation even easier. [38]

3.3 Design the work flow

After studying and researching our topic, we did not found a perfect solution that could be fit in our plans. So we combined the knowledge that we gained and some development techniques that we discovered to design our project.

The most of applications in our topic, where we discovered, have two similarities which we believe are not the ideal way to have a usable android application.

- The internet connection is necessary to execute the operations of the application.
- Using SPARQL queries in Android devices is a long time process.

Basically, the common weaknesses that we found were speed and the necessity of internet resource. So, in order to solve these issues, we developed the following architecture:

In our application, we use four different OWL ontologies. The basic mechanism to have the ontology data is the application to do (for each ontology) HTTP request in a remote server only once. The response request is the targeted ontology. After a successful response, all ontology data stored at the local memory of the device. So, if we want in the future to explore the same ontology it is not necessary to have an internet connection. Only at first time, when the user visits ontology the application is using internet connection.

The second part of our target is to solve the issue of speed. As we discussed at previous works, the use of SPARQL queries at mobile devices is not so wise solution because is a long time process. The android devices compared with desktop PCs have shorter memory and computing power.

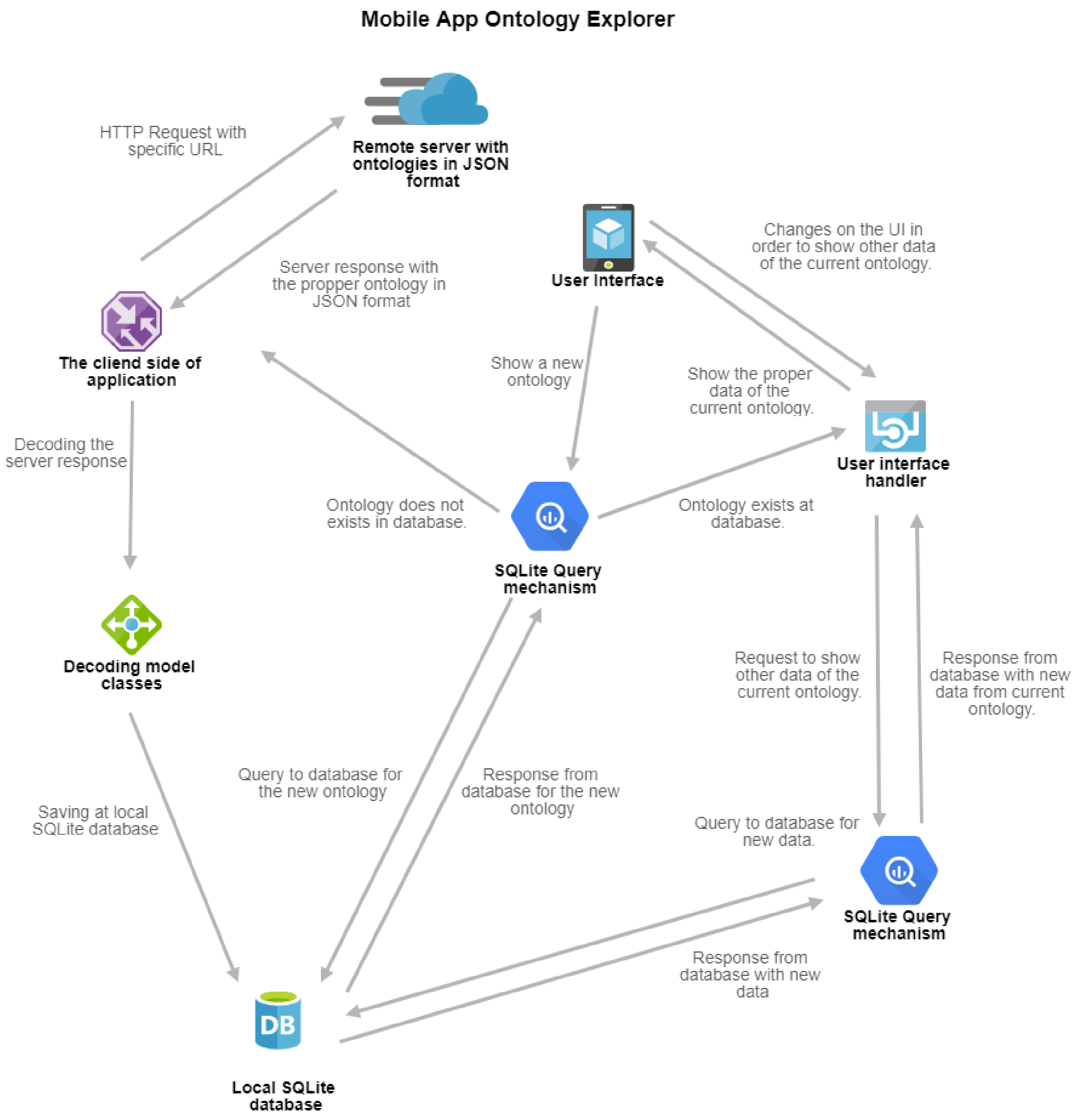
Considering that, we decide to use a familiar with android devices technology, a SQLite database. SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day. [39]

After the successful HTTP response from a remote server, the application gets the ontology data in JSON form. The application decodes this data and stored them into a local SQLite database. After this operation, we can gain any information about ontology by executing SQLite queries. In this way, the application can provide fast results to the user.

To sum up, we get the ontology data, we store them and we can explore them. The final step in order to complete the application is how these data could be displayed and makes a user-friendly environment for exploring the ontology.

Fortunately, Android has some helpful frameworks to build an excellent user interface. So, we used some of these and adapted them to our application.

The below graph depicts the workflow of our application. It sums up all operations that we describe above.


Mobile App Ontology Explorer
Remote server with ontologies in JSON format
HTTP Request with specific URL
Server response with the propper ontology in JSON format
The cliend side of application
Decoding the server response
Decoding model classes
Saving at local SQLite database
Local SQLite database
User Interface
Show a new ontology
Changes on the UI in order to show other data of the current ontology.
Show the proper data of the current ontology.
User interface handler
Ontology does not exists in database.
SQLite Query mechanism
Ontology exists at database.
Query to database for the new ontology
Response from database for the new ontology
Request to show other data of the current ontology.
Response from database with new data from current ontology.
Query to database for new data.
SQLite Query mechanism
Response from database with new data

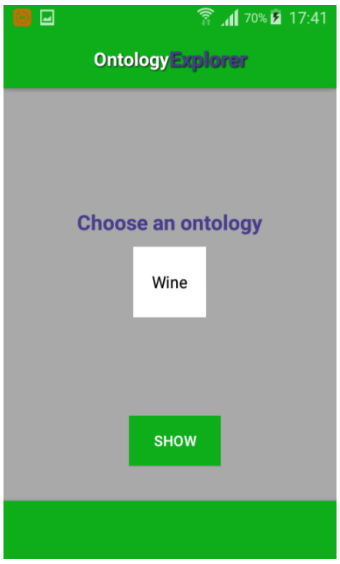
3.1 The workflow of Ontology Explorer application.

3.4 Screens

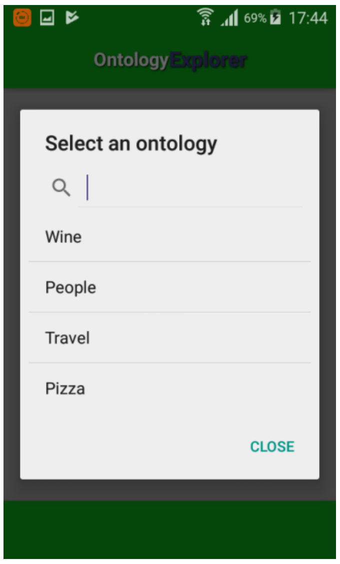
In this chapter demonstrates the application from the user aspect. We present the screens of the application and the operations of each one. The total main screens are three, moreover exists some dialogs and pop-up messages.

3.4.1 First screen

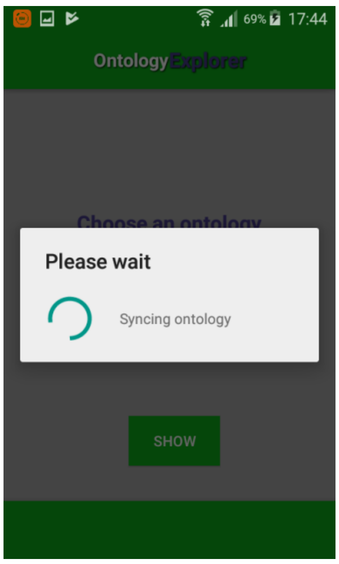
The main screen of the application, the user could choose an ontology (Wine, People, Travel, Pizza) from a dialog and view it by pressing the "SHOW" button. Consider that, if it is the first time which the user tries to explore a specific ontology, the application makes an HTTP request and shows an informative dialog to the user for this operation.



3.2 Main screen.



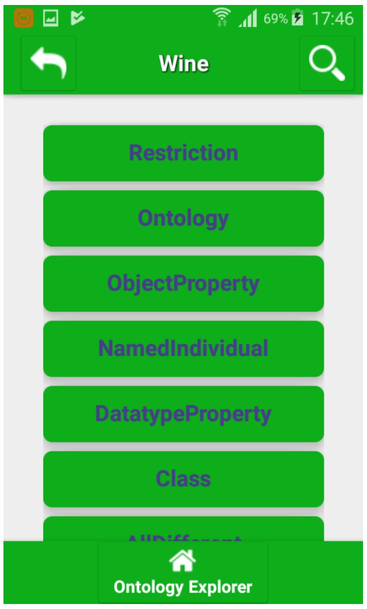
3.3 Ontology dialog.



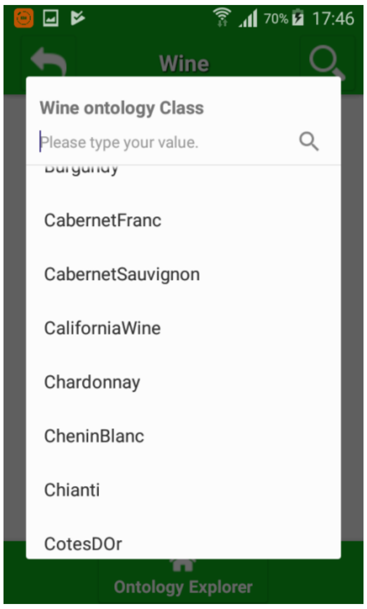
3.4 Syncing Ontology.

3.4.2 Second screen

Here exist the basic components of the ontology (Classes, Properties, Individuals, etc…) into a list. User can explore them by clicking the specific item of the list. By clicking an item, is showing a dialog with an object list. For example, if the user chooses the "Class" item application shows a list dialog with all classes of the ontology.

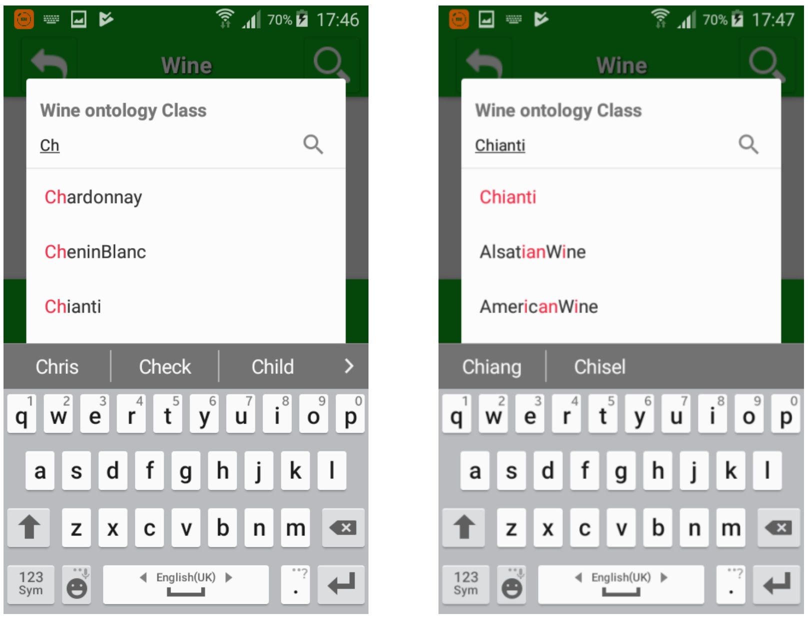


3.5 Second screen.



3.6 List dialog with classes.

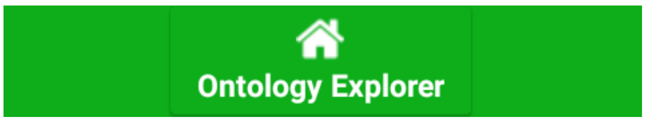
The user could search the item that he wants by scrolling up or down the list dialog. Moreover, the user can type the name of the item that he wants at the search bar of the dialog. The application will make a search and will show the results.



3.7 Searching a class by using the search bar.

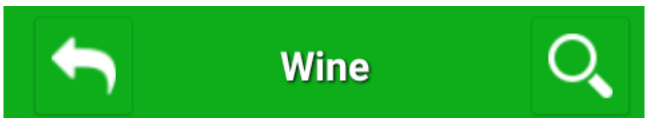
In order to explore the components of a specific item (for example the class Chianti), the user should click on the name of it. This event will trigger the next screen.

Except for the item list, this screen has more components. The first is a Bottom menu with a home button. By clicking this button, the application will go to the main screen; here user could choose another ontology to explore.

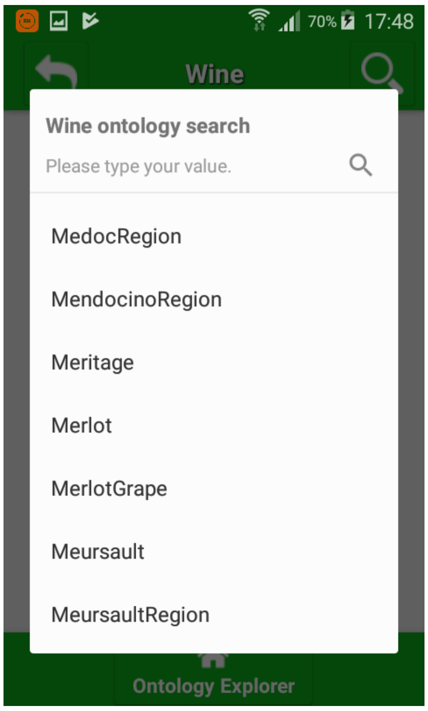


3.8 Bottom menu of first screen.

Furthermore has a top screen toolbar that includes the title of current ontology, a search button with all items of the ontology in a list format and a back arrow which leads to the previous screen.



3.9 Toolbar of first screen.



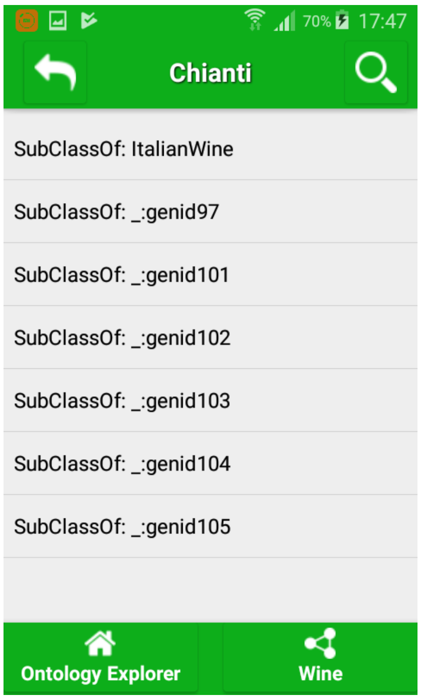
Wine ontology search
Please type your value.

MedocRegion
MendocinoRegion
Meritage
Merlot
MerlotGrape
Meursault
MeursaultRegion

3.10 List dialog with ontology items.

3.4.3 Third screen

This screen has a dynamical list of components from the chosen item, for example, if we choose the Chianti item from the previous screen this list will be filled with the data of Chianti class, “SubClassOf: ItalianWine” etc.

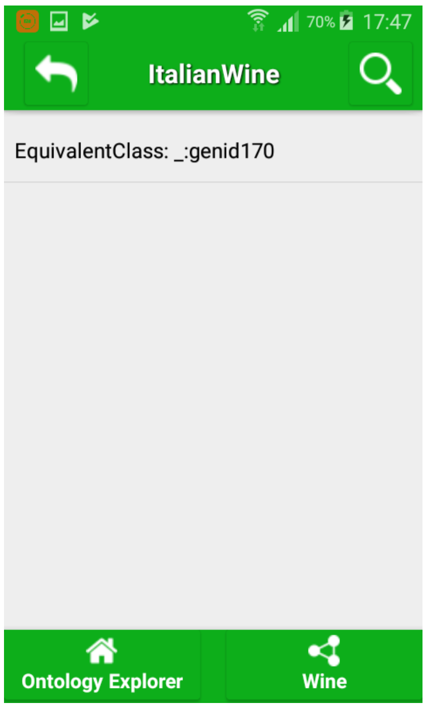


SubClassOf: ItalianWine
SubClassOf: _:genid97
SubClassOf: _:genid101
SubClassOf: _:genid102
SubClassOf: _:genid103
SubClassOf: _:genid104
SubClassOf: _:genid105

3.11 Third screen.

If a user wants to search deeper to ontology, is needed to click a new item. In our example had chosen Chianti class and saw their data. One of these data is

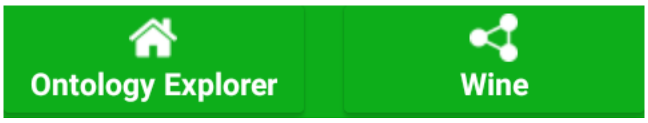
“ItalianWine” if the user wants to see the data of this component should click on it. By clicking a new item, the list of the screen will be refreshed and will show the data of new item, in our example user clicks “ItalianWine” item and the list shows “EquivalentClass:_:genid170. This operation could continue by clicking the new items.



EquivalentClass: _:genid170

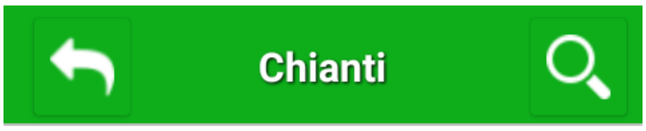
3.12 The refreshing list after user click.

Moreover, this screen includes a bottom menu with two buttons. The first button leads to the main activity in order to choose a different ontology. The second button leads at the second screen which exists the basic components of current ontology, as we describe earlier.



3.13 Bottom menu of third screen.

The last item of this screen is a top screen toolbar that shows the name of the current item, has a search button in order to search an item instantly and has a back arrow which leads to previous ontology item or previous screen.



3.14 Tool bar of third screen.

3.5 Back-end operations

After a demonstration of application from user aspect, is vital thing to explain the back end process for each feature and what tools we used to develop them.

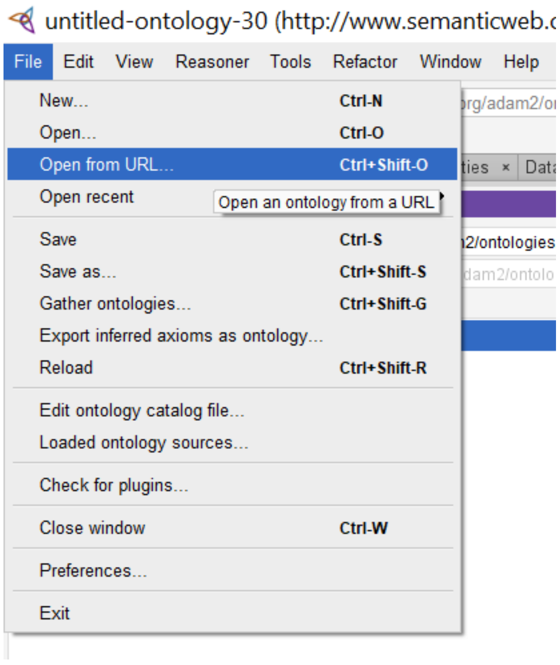
In this chapter, we describe all the lifecycle of our application. Which includes HTTP requests from the mobile to a remote server in order to get the ontology data, local storing of these data to the device and the depiction to the user interface.

3.5.1 Ontologies and JSON format.

In order to make these ontologies capable to connect remotely, we use a specific format for the ontology data, the JSON format. The JavaScript Object Notation (JSON) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types (or any other serializable value). [40]

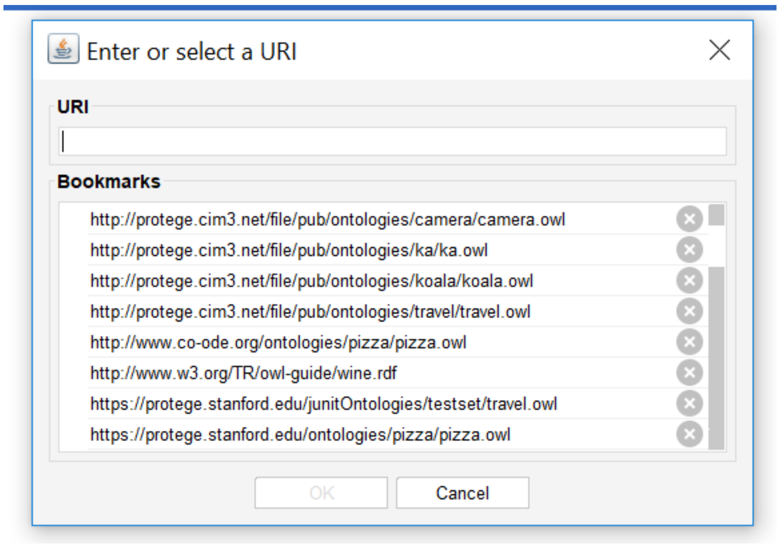
The first step is to make the ontologies at JSON format. For this operation, we used the protégé, which is a desktop software to edit ontologies. We demonstrate step by step the settings that we followed:

1. Download and install the Protégé.
   Protégé is a popular open source project, furthermore, it is free. So, we downloaded the latest version from the official site [22]. Following the instructions from site, we easy done the installation to our pc.
2. Open the ontology to protégé
   The next step is to open the ontologies to Protégé we did it by this path: File -> Open from URL


untitled-ontology-30 (http://www.semanticweb.c
File
Edit
View
Reasoner
Tools
Refactor
Window
Help
New... Ctrl-N
Open... Ctrl-O
Open from URL... Ctrl+Shift-O
Open recent
Open an ontology from a URL
Save Ctrl-S
Save as... Ctrl+Shift-S
Gather ontologies... Ctrl+Shift-G
Export inferred axioms as ontology...
Reload Ctrl+Shift-R
Edit ontology catalog file...
Loaded ontology sources...
Check for plugins...
Close window Ctrl-W
Preferences...
Exit

3.15 Open ontology from URL at Protégé.

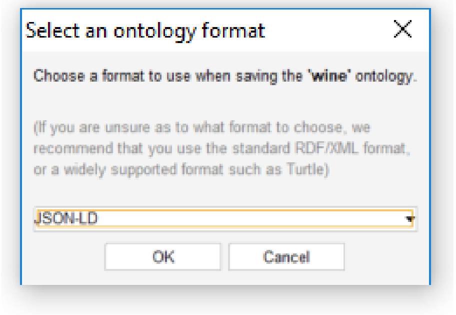
After this, a dialog shows with some ontologies where exist at Web. We choose one by one the ontologies (Wine, People, Travel, and Pizza).



http://protege.cim3.net/file/pub/ontologies/camera/camera.owl
http://protege.cim3.net/file/pub/ontologies/ka/ka.owl
http://protege.cim3.net/file/pub/ontologies/koala/koala.owl
http://protege.cim3.net/file/pub/ontologies/travel/travel.owl
http://www.co-ode.org/ontologies/pizza/pizza.owl
http://www.w3.org/TR/owl-guide/wine.rdf
https://protege.stanford.edu/junitOntologies/testset/travel.owl
https://protege.stanford.edu/ontologies/pizza/pizza.owl

3.16 Choosing an ontology.

3. Save the ontology with JSON format.
   Protégé downloaded the chosen ontology from the web and load it locally. After this operation, the next step was to save the ontology with JSON format.



Choose a format to use when saving the '**wine**' ontology.

(If you are unsure as to what format to choose, we recommend that you use the standard RDF/XML format, or a widely supported format such as Turtle)

JSON-LD

3.17 Saving an ontology with JSON format.

We continued these steps for each ontology and finally, we had four JSON files. We demonstrate the "Chianti" class of Wine ontology, how to depicts in a JSON file.

```
"@id" : "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Chianti",
  "@type" : [ "http://www.w3.org/2002/07/owl#Class" ],
  "http://www.w3.org/2000/01/rdf-schema#subClassOf" : [ {
    "@id" : "http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#ItalianWine"
  }, {
    "@id" : "_:genid97"
  }, {
    "@id" : "_:genid101"
  }, {
    "@id" : "_:genid102"
  }, {
    "@id" : "_:genid103"
  }, {
    "@id" : "_:genid104"
  }, {
    "@id" : "_:genid105"
  } ]
},
```
3.18 “Chianti” class with JSON format.

3.5.2 The remote server

We made the ontologies with JSON format, now we need a server to store them. For this operation, we used a helpful tool for the Myjson web application. [41]

Myjson is a place to storing configuration file for client-side models. So, we transferred the ontologies with JSON format into the Myjson application server. Then the application server provided a URL for each ontology. Using this URL we have access to ontology data from a mobile device. The provided URLs are the following:

- Wine ontology - https://api.myjson.com/bins/14u1dt
- People ontology-https://api.myjson.com/bins/ulhfy
- Travel ontology- https://api.myjson.com/bins/1ecqmm
- Pizza ontology-https://api.myjson.com/bins/uafji

3.5.3 HTTP request from mobile.

The next step was to construct a client mechanism at the mobile device, we developed this feature by using a powerful library “Retrofit”. [42]

“Retrofit” library provides all the necessary methods for HTTP requests through a very helpful Java interface.

Firstly, we included the “Retrofit” library to our project. To solve this we use the Grandle file. Grandle file is an advanced build toolkit to automate and manage the build process, is used by Android Studio. [43]

Using Gradle file the only thing that needs to include the “Retrofit” library is to add the proper externalNativeBuild block. This is a link that leads to the library repository. After this, the Gradle is syncing the files of the library with the Android project.

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'
compile 'com.squareup.retrofit2:converter-gson:2.2.0'
```
3.19 The external link of “Retrofit” library.

Using all necessary tools of “Retrofit” library we developed a method to making HTTP calls at the remote server. This method takes the specific URL of each ontology as a parameter.

```
public void callOntology(String query){

    Retrofit retrofit = new
Retrofit.Builder().baseUrl("https://api.myjson.com/").

        addConverterFactory(GsonConverterFactory.create()).build
    final Api api = retrofit.create(Api.class);

    call2 = api.getWineOntology(query);
    call2.enqueue(new Callback<JsonArray>() {
        @Override
        public void onResponse(Call<JsonArray> call, Response<JsonArray>
response) {
            decodingResponse = new DecodingResponse(MainActivity.this,
                                   String.valueOf(response.body()),
                                   ontologySelection);
            progressDialog.dismiss();
            Intent myIntent = new Intent(MainActivity.this,
OntologyData.class);
            startActivity(myIntent);
        }

        @Override
        public void onFailure(Call<JsonArray> call, Throwable t) {
            Toast.makeText(getApplicationContext(),
                 "No internet connection",Toast.LENGTH_SHORT).show();

            Log.d("TAG_ERROR", t.getMessage());
        }
    });
}
```

3.20 The method to get ontology data from server

3.5.4 Decoding the response

After a successful call to server, we need to decode the response. The server response is all the ontology in JSON format, as we described at the previous chapter. For JSON decoding we developed a model that makes the proper operations to gain only the usable information of ontology.

For decode, the server response, each component of ontology needs a different model (Class model, Individual model, Property model, etc.). The model is capable to decode the response and save the necessary information to a local SQLite database. Using this technique the HTTP call will become only at first time when the user visits the ontology.

```
public void decodingProperty(Context context,final String
propertyStr,Integer objectID){

    MyAppDatabase myAppDatabase =
MyAppDatabase.getAppDatabase(context);

    try {
        JSONArray propertyArray = new JSONArray(propertyStr);
        for(int j=0; j<propertyArray.length();j++){
            JSONObject jo2 = propertyArray.getJSONObject(j);
            String propertyHelp = jo2.getString("@id");
```

```
                if(propertyHelp.contains("#")){
                    propertyHelp = gFunctions.
                                seperatedResponse(jo2.getString("@id"));
                }

                Property property = new Property();
                property.setObjectId(objectID);
                property.setProperty(propertyHelp);

                myAppDatabase.MyDao().addProperty(property);

                ObjectData objectData = new ObjectData();
                objectData.setObjectId(objectID);
                objectData.setModel("Property");
                objectData.setValue(propertyHelp);
                myAppDatabase.MyDao().addObjectData(objectData);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
}
```

3.21 The method to decode properties.

3.5.5 Local Storing

To handle SQLite database operations we used another powerful Android library the “Room”. This library provides an abstraction layer over SQLite for more robust database access. [44]

We used the Grandle file in order to include it at our project, in the same way as we did before with “Retrofit” library.

```
implementation 'android.arch.persistence.room:runtime:1.0.0'
annotationProcessor 'android.arch.persistence.room:compiler:1.0.0'
```

3.22 The external link of “Room” library.

Using the tools that this library provides we developed our database schema and the necessary SQLite queries for our project.

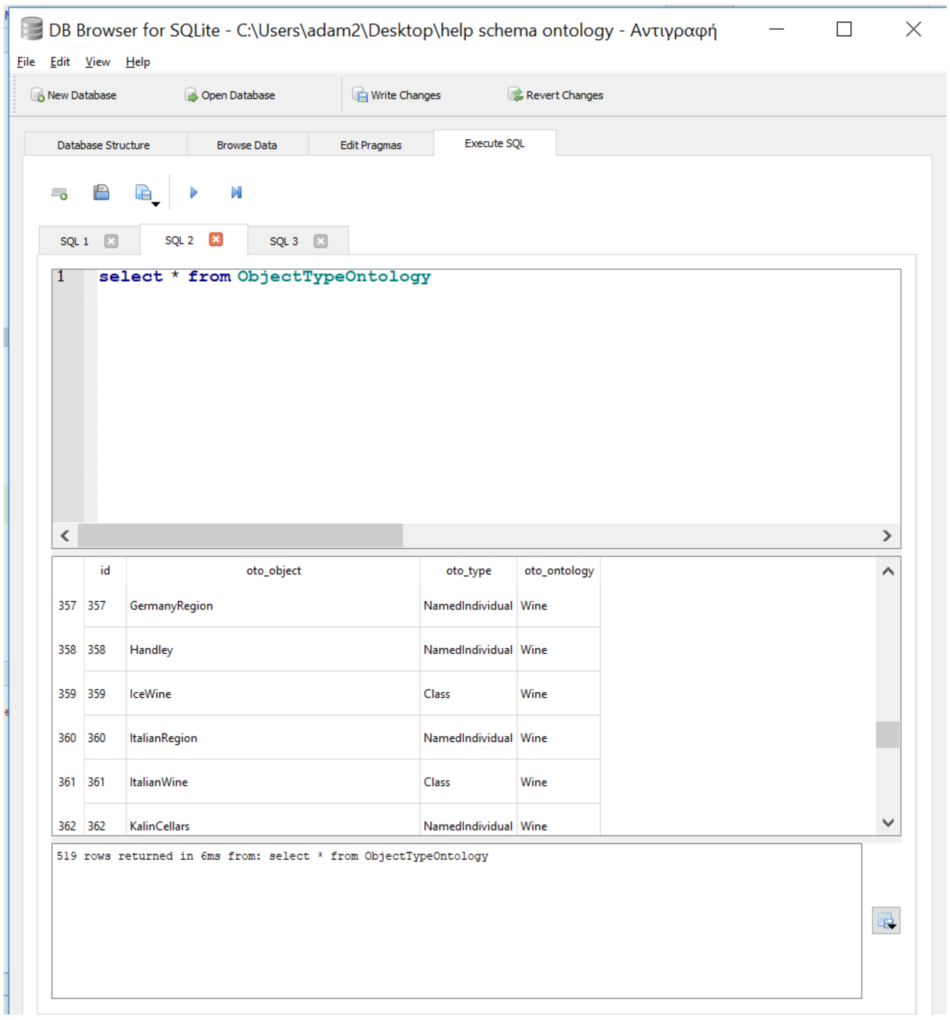
```
@Insert(onConflict = IGNORE)
public long addOTO(ObjectTypeOntology oto);

@Query("select * from ObjectTypeOntology where oto_ontology
=:ontology group by oto_type")
public List<ObjectTypeOntology> getOntologyTypes(String ontology);

@Query("select oto_object from ObjectTypeOntology where oto_type
=:type and oto_ontology=:ontology")
public List<String> getObjectsFromType(String type,String ontology);
```

3.23 Some SQLite queries in the project.

The application uses three tables for all operations (ObjectTypeOntology, ObjectData, and Ontology). We used the DB Browser for SQLite, which is a helpful tool in order to view the database data during the development. [45]

Follows a description of the fields for each table.

1. ObjectTypeOntology fields:
   - id - An auto-generate field in order to have a unique item for each row of the table.
   - oto_object - The name of the decoding object.
   - oto_type - The type of each object.
   - oto_ontology - The ontology of each object.



```
select * from ObjectTypeOntology
```

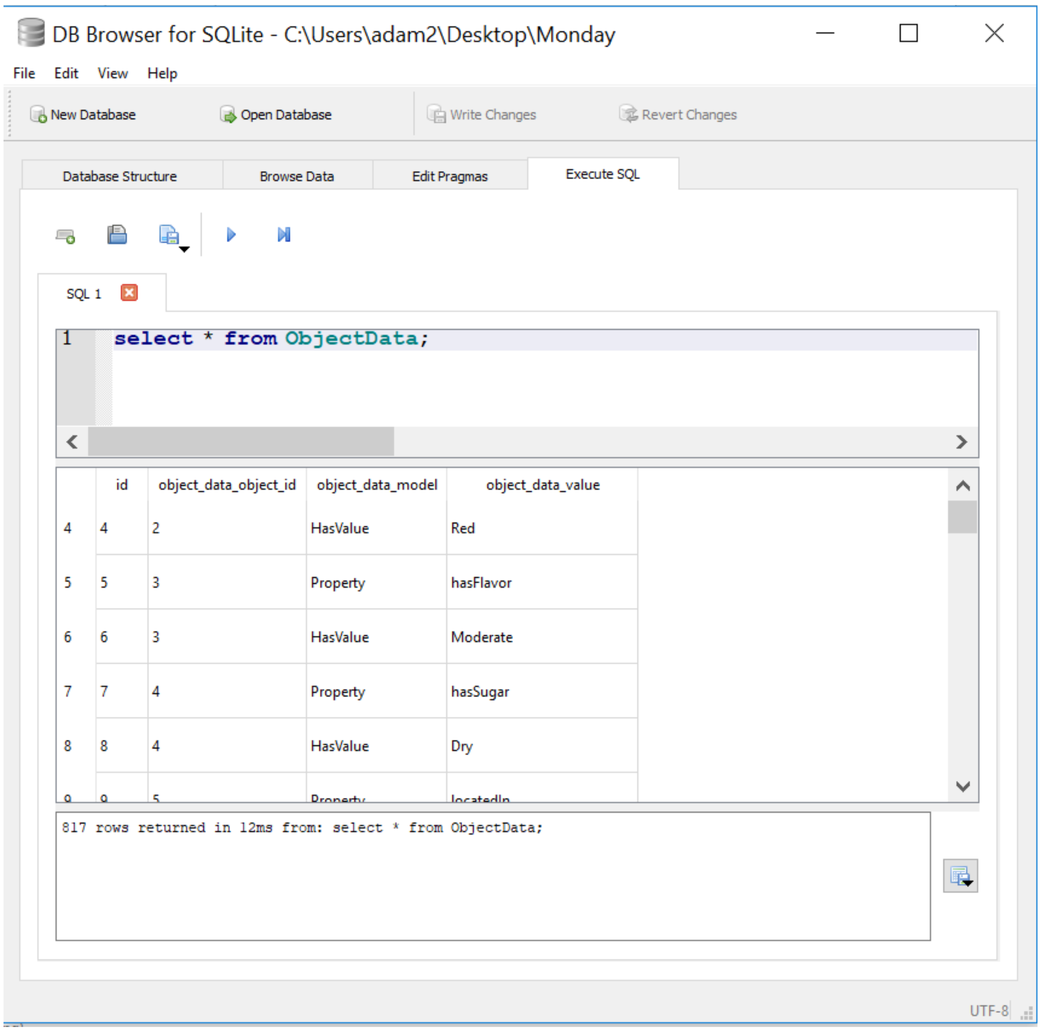
	id	oto_object	oto_type	oto_ontology
357	357	GermanyRegion	NamedIndividual	Wine
358	358	Handley	NamedIndividual	Wine
359	359	IceWine	Class	Wine
360	360	ItalianRegion	NamedIndividual	Wine
361	361	ItalianWine	Class	Wine
362	362	KalinCellars	NamedIndividual	Wine

```
519 rows returned in 6ms from: select * from ObjectTypeOntology
```

3.24 The ObjectTypeOntology table.

2. ObjectData fields:
   - id- An auto-generate field in order to have a unique item for each row of the table.

- object_data_object_id - The object id from table ObjectTypeOntology.
- object_data_model - The model of the object.
- object_data_value - The value of the object.



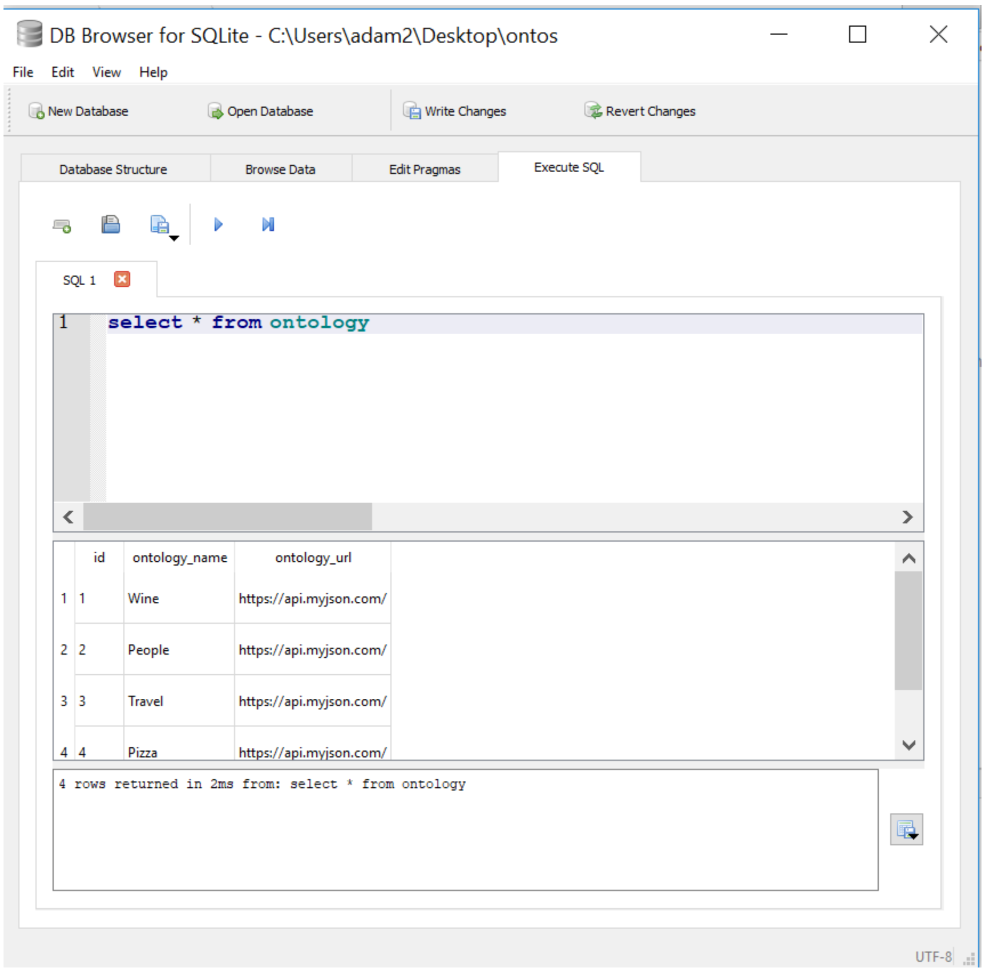
```
select * from ObjectData;
```

	id	object_data_object_id	object_data_model	object_data_value
4	4	2	HasValue	Red
5	5	3	Property	hasFlavor
6	6	3	HasValue	Moderate
7	7	4	Property	hasSugar
8	8	4	HasValue	Dry
9	9	5	Property	locatedIn

```
817 rows returned in 12ms from: select * from ObjectData;
```

3.25 The ObjectData table.

3. Ontology fields
   - id - An auto-generate field in order to have a unique item for each row of the table.
   - ontology_name – the name of each ontology.
   - ontology_url – the targeting URL from the remote server of each ontology.



```
select * from ontology
```

	id	ontology_name	ontology_url
1	1	Wine	https://api.myjson.com/
2	2	People	https://api.myjson.com/
3	3	Travel	https://api.myjson.com/
4	4	Pizza	https://api.myjson.com/

```
4 rows returned in 2ms from: select * from ontology
```

3.26 The Ontology table.

3.5.6 User Interface

In this chapter, we will describe the components that we used in order to build the user interface of the application. Considering that we design it for mobile devices, so depending on the screen size the interface should be different than desktop applications.

Android provides to developer a variety of pre-built UI components that allow building the graphical user interface for the app. [46]

We describe all the user interface components where we used in our project and the operation of each one, at the bellowing table.

Component name	Component description
ConsraintLayout	Allows to position and size UI elements in a flexible way. [47]
ToolBar	A component within the activity that might display other interactive items. [48]
Button	An element which the user can tap in order to perform an action. [48]
BottomNavigationView	Represents a bottom navigation bar for an application. [49]
TextView	An element which displays a text to the user. [50]
SearchableSpinner	A drop-down list which user can select one option. [51]
RecyclerView	A flexible view in order to provide a limited window into a large data set. [52]
CardView	A frame with a rounded corner background and shadow. [53]
ImageView	This element is used to display an image resource [54]
LinearLayout	A view group which aligns all children in a single direction (vertically or horizontally). [55]
ScrollView	A view group which allows the view placed within it to be scrolled. [56]
ListView	A view of a vertical-scrollable collection of items. [57]
Dialog	A small window which includes a message or prompts the user to make a decision. [58]

Bellow we demonstrate the source code of material_card.xml file; this file includes a CardView, a LinearLayout and a TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardView"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    app:cardCornerRadius="8dp"
    app:cardElevation="8dp">

    <LinearLayout
        android:id="@+id/types_linear_layout"
        android:background="#0DAE1A"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:clickable="true"
        android:focusable="true"
        android:foreground="?attr/selectableItemBackground"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tv_type"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_gravity="center"
            android:layout_toEndOf="@+id/imageView_list"
            android:layout_toRightOf="@+id/imageView_list"
            android:layout_marginTop="10dp"
            android:text="TextView"
            android:textStyle="bold"
            android:textColor="#483D8B"
            android:textSize="20dp" />

    </LinearLayout>

</android.support.v7.widget.CardView>
```

3.27 material_card.xml file

3.6 Files and Classes

In this chapter we will demonstrate all application files, classes and their operations that we used in order to build our application. In our project we have 33 java classes and 4 XML files.

Name of Java class	Description of Java class
Api.class	Includes the method of the HTTP request to the remote server.
CardItem	Handling the items of RecyclerView list at the second screen.
CheckValues	Validation of server response and collection the existing fields of ontology.
DecodingResponse	Decoding of server response and saving the data to

	the local database.
GeneralFunctions	Includes support methods that are used at multiple classes (example separation of String).
MainActivity	The handler of all functions for the first screen.
MyAppDatabase	Includes the necessary methods for Initializing the database of the application.
MyDao	Includes all SQLite queries where the application is using.
ObjectData	The components of the ObjectData table.
ObjectDepiction	Handler of all functions for third screen.
ObjectTypeOntology	The components of ObjectTypeOntology table.
Ontology	The components of Ontology table.
OntologyData	The handler of all functions for the third screen.
PrefsHandler	Keeps the current ontology to a temporary file.
RecyclerTypeListAdapter	Adapts the items of RecyclerView list.
SampleSearchModel	Helps to search an object to current ontology.
Model classes	17 classes where decoding each component of the Ontology: AllValueFrom, Cardinality, DistrictMembers, Domain, EquivalentClass, HasValue, IntersectionOf, InverseOf, MaxCardinality, MinCardinality, OneOf, Property, Range, SomeValuesFrom, SubClassOf, SubPropertyOf, UnionOf.

Name of XML file	Description of XML file
activity_main.xml	The user interface of the first screen.
activity_ontology_data.xml	The user interface of the second screen.
activity_object_depiction.xml	The user interface of the third screen.
material_card	A support file which depicts each item from RecyclerView list at the second screen.

Chapter 4

Usability evaluation

4.1 Introduction

In this chapter, we demonstrate the experiments where we have done in order to evaluate the usability of Ontology Explorer Android application.

Firstly we created ten scenarios in order for the users to find some specific data from the ontologies. The users first tried to found the data to our application then at a desktop application Protégé.

Firstly we created ten scenarios in order for the users to find some specific data from the ontologies. The users first tried to found the data to our application then at a desktop application “Protégé”.

4.2 Scenarios

From Wine ontology the user should find the below data:

- Find the data type property of Wine ontology.
- Find the maker of ChiantiClassico wine.
- Find the color of Chianti wine.

From People ontology the user should find the below data:

- Find the intersections of woman class.
- Find what a giraffe eats.

From Travel ontology the user should find the below data:

- Find the data type properties of Travel ontology.
- Find the superclass of the Hotel.

From Pizza ontology the user should find the below data:

- Find the range of “has Ingredient” property.
- Find the toppings of Margherita Pizza.
- Find the toppings of American Pizza.

4.3 Experiment preparation

The experiments became by using specific devices:

- For Android application we used an MLS smartphone:
  - Model: MLS F5
  - Operating system: Android 7.0
  - RAM: 2GB
  - CPU: Cortex-A7, 4 cores, 1.2GHz
- For Protégé application we used an ASUS ultrabook:
  - Model: Asus ZenBook 14΄
  - Operating system: Windows 10
  - RAM: 8GB
  - CPU: Intel Core i5, 2.5GHz

Before the experiment, we demonstrated both applications (Ontology Explorer, Protégé). When the user had understood the basic operations for each software the experiment begins. The time of the demonstration was between 5 to 10 minutes, depending on the user experience. Mention that some users are familiar with the Protégé application.

During the experiment, the user tries to answer the questions where we described before. We did not stop the experiment until the user answers correctly all of the questions.

4.4 Usability evaluation to the Android application

All the users tested first the Android application. The four ontologies are synced before the experiment in a local database in order for the users to not spending time during the experiment for this operation. We want to know only the needed time that a user to find some results at the Android application. The experiment results exist at the following table:

User	Time
Petros`	07:57.70 sec
Eirene	12:22.20 sec
Eythimis	05:49.65sec
Manos	06:10.08 sec
Giannis	05:36.63 sec
Maria	14:53.02 sec
Kostas	15:10.34 sec
Hlias	08:23.20 sec
Dimitris	08:01.55 sec
Nikos	09:20.01 sec

4.5 Comparison the results with Protégé

The next step was the user to answer the same questions by using the desktop application Protégé. We have stored locally to the laptop all ontologies that we needed in order to not spending time when we loaded them to Protégé, like previous in Android application. We present the comparison results at the following table:

User	Android app time	Protégé time	Familiar with Protégé
Petros	7:57.70 sec	11:39.50 sec	No
Eirene	12:22.20 sec	18:09.05 sec	No
Eythimis	05:49.65sec	09:24.18 sec	Yes
Manos	06:10.08 sec	12:32.05 sec	Yes
Giannis	05:36.63 sec	15:02.41 sec	No
Maria	14:53.02 sec	20:31.22 sec	No
Kostas	15:10.34 sec	19:42.55 sec	No
Hlias	08:23.20 sec	12:50.01 sec	Yes
Dimitris	08:01.55 sec	13:25.12 sec	No
Nikos	09:20.01 sec	13:53.16 sec	No

Although the experiment sample is not so big we could conclude that all users find the results at Android application faster. Of course in Protégé the user could make more operations like developing a new ontology or make custom SPARQL queries. However, in these experiments, we studied how fast a user can explore an ontology. In this sample, the Android application that we developed is faster than Protégé.

After the experiment, some users said that one of the vital benefits in Ontology Explorer application is the simple UI. Protégé has more buttons and tags in the UI because, as we said before, has more features.

4.6 Testing the speed performance

After the user interaction experiments, we tested the application in multiple devices. In order to validate if the application is stable at different versions of the Android operating system.

Moreover, another useful aspect is to know the time that application needs to make the synchronization operation: HTTP request to the remote server, decode the response, storing it at the local database. We used the following devices in our experiment:

Device	Model	Op. System	RAM	CPU	Type
Device 1	MLS F5	Android 7.0	2GB	Quad core, 1.2GHz	Smartphone
Device 2	Sumsung Prime	Android 5.0.2	1GB	Quad core, 1.2GHz	Smartphone
Device 3	Sumsung Galaxy TAB pro	Android 4.4.2	2GB	Quad core, 1.3 GHz	Tablet
Device 4	Sumsung J3	Android 8.1	2GB	Quad core, 1.4GHz	Smartphone

We installed the application at different devices and we timed the decoding operation. The results are demonstrated at the following table:

Device	Wine	People	Travel	Pizza
Device 1	18.87 sec	5.57 sec	4.80 sec	11.77 sec
Device 2	21.44 sec	5.44 sec	4.06 sec	14.15 sec
Device 3	20.54 sec	6.51 sec	4.28 sec	15.43 sec
Device 4	21.18 sec	5.55 sec	3.72 sec	13.88 sec

We observe that the time to synchronize all ontology data from the remote server depends on ontology size. The Wine ontology has more data than People ontology so needs more time for synchronization.

Furthermore, the synchronization operation does not need much time to conclude. The longer time in our experiment is 21.44 sec when the device 2 is synchronizing the Wine ontology.

It is a logical time value because this operation executes only at first time when the user visits a specific ontology. When this operation finished the ontology exists locally at the mobile device. So, if the user wants to see again the ontology the execution will be instant.

Chapter 5

Conclusion and future work

5.1 General conclusion

OWL ontologies are an interesting and helpful way to depict a dataset. It is a general format which in the future could have more and more usages. The major benefit of it is that is easy to read from people, so it is simple to make a search on it.

Many desktop applications exist in order to handle OWL ontologies. However, at mobile devices all related applications are in experimental stage. In order to provide a solution of this issue we developed the Ontology Explorer. It is an innovating Android application that the user could explore OWL ontologies.

We achieve this target by using a combination of tested technologies from Android platform and new approach for handling OWL ontologies. Through some experiments of usability evaluation of application we conclude that is a helpful and practical useful.

However, the application could extend the features and be even more useful.

5.2 Future work

As we described above, considering the experiments the application is practically useful. However, if extends some feature could be even more helpful for the users. We demonstrate some suggestions for future work:

- One of the future features that the Ontology Explorer application could have is to sync a new custom ontology from the user. Now the application has four OWL ontologies that triggered by internal URLs. In this basis, the user will load a new ontology by typing the proper URL at Ontology Explorer application.
- Another feature that could have the application is the operation to modify an ontology. After syncing, the user could create/delete a component of the ontology (class, property, etc.)
- The last suggestion for future work is to develop a tablet version. The Ontology Explorer application runs in tablets, however, it uses the same UI as the smartphones. As a result, is that we have a lot of not used space because the screen of the tablet is bigger than a smartphone. So, a version which fits in the tablet screen will be very useful.

References

[1] Yus, R., Bobed, C., Esteban, G., Bobillo, F., & Mena, E. (2013, July). Android goes Semantic: DL Reasoners on Smartphones. In *Ore* (pp. 46-52).

[2] Semantic reasoner. (2019, May 12). Retrieved from https://en.wikipedia.org/wiki/Semantic_reasoner

[3] Dalvik (software). (2019, May 05). Retrieved from https://en.wikipedia.org/wiki/Dalvik_(software)

[4] Bobed, C., Bobillo, F., Yus, R., Esteban, G., & Mena, E. (2014). Android Went Semantic: Time for Evaluation. In *ORE* (pp. 23-29).

[5] Becker, C., & Bizer, C. (2008). DBpedia Mobile: A Location-Enabled Linked Data Browser. *Ldow*, *369*, 2008.

[6] d'Aquin, M., Nikolov, A., & Motta, E. (2011). Building SPARQL-enabled applications with android devices.

[7] Roşoiu, M., David, J., & Euzenat, J. (2012, May 27). A Linked Data Framework for Android. Retrieved from https://link.springer.com/chapter/10.1007/978-3-662-46641-4_15

[8] Introduction to Activities | Android Developers. (n.d.). Retrieved from https://developer.android.com/guide/components/activities/intro-activities

[9] Android Developers. (2019). *Intent | Android Developers*. (n.d.). Retrieved from https://developer.android.com/reference/android/content/Intent

[10] Services overview | Android Developers. (n.d.). Retrieved from https://developer.android.com/guide/components/services

[11] *BroadcastReceiver | Android Developers*. (n.d.). Retrieved from https://developer.android.com/reference/android/content/BroadcastReceiver

[12] *Content providers | Android Developers*. (n.d.). Retrieved from *https://developer.android.com/guide/topics/providers/content-providers.*

[13] JS Foundation. (n.d.). A Touch-Optimized Web Framework. Retrieved from http://jquerymobile.com/

[14] B. D. (2011, October 4). Displaying SPARQL results on a mobile phone. Retrieved from http://www.snee.com/bobdc.blog/2011/10/displaying-sparql-results-on-a.html

[15] Viswanathan. (2019, May 21). Should You Develop a Native App or a Web App? Retrieved from https://www.lifewire.com/native-apps-vs-web-apps-2373133

[16] Google Code Archive - Long-term storage for Google Code Project Hosting. (n.d.). Retrieved from https://code.google.com/archive/p/androjena/downloads

[17] Google Code Archive - Long-term storage for Google Code Project Hosting. (n.d.). Retrieved from https://code.google.com/archive/p/androjena/wikis/ARQoid.wiki

[18] Read, D. (2011, December 01). Using ARQoid for Android-based SPARQL Query Execution. Retrieved from https://monead.com/blog/?p=1420

[19] Sparql Droid - Apps on Google Play. (n.d.). Retrieved from https://play.google.com/store/apps/details?id=com.monead.semantic.android.sparql

[20] Unionpedia - Apps on Google Play. (n.d.). Retrieved from https://play.google.com/store/apps/details?id=org.unionpedia

[21] PascoLink - Apps on Google Play. (n.d.). Retrieved from https://play.google.com/store/apps/details?id=forth.ics.pascolink

[22] Stanford Center for Biomedical Informatics Research. (n.d.). A free, open-source ontology editor and framework for building intelligent systems. Retrieved from https://protege.stanford.edu/

[23] NeOn Wiki. (n.d.). Retrieved from http://neon-toolkit.org/wiki/Main_Page.html

[24] SWOOP. (n.d.). Retrieved from https://www.w3.org/2001/sw/wiki/SWOOP

[25] TopBraid Composer - Maestro Edition. (n.d.). Retrieved from https://www.topquadrant.com/products/topbraid-composer/

[26] Vivo-Project. (2019, May 08). Vivo-project/Vitro. Retrieved from https://github.com/vivo-project/Vitro

[27] Knoodl. (n.d.). Retrieved from http://knoodl.com/

[28] Anzo. (n.d.). Retrieved from https://www.w3.org/2001/sw/wiki/Anzo

[29] Finally, an easy wayto work with ontologies. (n.d.). Retrieved from http://owlgred.lumii.lv/

[30] Cognitum Company. (n.d.). Data-Driven Digital Transformation with A.I. Retrieved from https://www.cognitum.eu/Semantics/FluentEditor/

[31] Semantic Turkey. (n.d.). Retrieved from http://semanticturkey.uniroma2.it/

[32] VocBench. (n.d.). Retrieved from http://vocbench.uniroma2.it/

[33] A Guide to Mobile App Development: Web vs. Native vs. Hybrid. Retrieved from https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/

[34] Mobile operating system. (2019, June 25). Retrieved from https://en.wikipedia.org/wiki/Mobile_operating_system

[35] Devices. (n.d.). Retrieved from https://www.android.com/devices/

[36] Mobile Operating System Market Share Worldwide. (n.d.). Retrieved from http://gs.statcounter.com/os-market-share/mobile/worldwide

[38] Top 20 Tools for Android Development. (n.d.). Retrieved from https://www.altexsoft.com/blog/engineering/top-20-tools-for-android-development/

[37] Download Android Studio and SDK tools. (n.d.). Retrieved from https://developer.android.com/studio/?gclid=Cj0KCQjw3PLnBRCpARIsAKaUbgt3dg-Qz9yDrVl8X1SZOtCJxU3ZhRuHqAV6Ai3n4AFSuSvtZsOVEYAaAsxfEALw_wcB

[39] (n.d.). Retrieved from https://www.sqlite.org/index.html

[40] JSON. (2019, June 27). Retrieved from https://en.wikipedia.org/wiki/JSON

[41] { } myjson. (n.d.). Retrieved from http://myjson.com/about

[42] Retrofit. (n.d.). Retrieved from https://square.github.io/retrofit/

[43] Configure your build | Android Developers. (n.d.). Retrieved from https://developer.android.com/studio/build

[44] Room| Android Developers. (n.d.). Retrieved from https://developer.android.com/jetpack/androidx/releases/room

[45] DB Browser for SQLite. (n.d.). Retrieved from https://sqlitebrowser.org/

[46] User Interface & Navigation | Android Developers. (2019). Retrieved from https://developer.android.com/guide/topics/ui

[47] ConstraintLayout | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/support/constraint/ConstraintLayout

[48] ActionBar | Android Developers. (2019). Retrieved from https://developer.android.com/reference/androidx/appcompat/app/ActionBar.html

[48] Button | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/widget/Button

[49] BottomNavigationView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/support/design/widget/BottomNavigationView

[50] TextView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/widget/TextView

[51] A Searchable Spinner in Android ~ Life News. (2019). Retrieved from https://life-news.blog/2018/11/16/a-searchable-spinner-in-android/

[52] RecyclerView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/support/v7/widget/RecyclerView

[53] CardView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/support/v7/widget/CardView

[54] ImageView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/widget/ImageView

[55] Linear Layout | Android Developers. (2019). Retrieved from https://developer.android.com/guide/topics/ui/layout/linear

[56] ScrollView | Android Developers. (2019). Retrieved from https://developer.android.com/reference/android/widget/ScrollView

[57] Linear Layout | Android Developers. (2019). Retrieved from https://developer.android.com/guide/topics/ui/layout/linear

[58] Dialogs | Android Developers. (2019). Retrieved from https://developer.android.com/guide/topics/ui/dialogs