Caching for Client-side Browsing of SPARQL Endpoints

Vaggelis Kalligiannakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes Campus, GR-70013 Heraklion, Crete, Greece

Thesis Advisor: Associate Prof. *Yannis Tzitzikas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology – Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Caching for Client-side Browsing of SPARQL Endpoints

Thesis submitted by
Vaggelis Kalligiannakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Vaggelis Kalligiannakis

Committee approvals: _____
Yannis Tzitzikas
Associate Professor, Thesis Supervisor

_____
Dimitris Plexousakis
Professor, Committee Member

_____
Irini Fundulaki
Principal Researcher, Committee Member

Departmental approval: _____
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, July 2016

Abstract

More and more data are published according to the principles of Linked Data and are accessible through SPARQL endpoints. Since data browsing through Web browsers is not always supported and given the increasing use of smart devices (phones, tablets) that are equipped with web browsers, in this thesis we elaborate on how we can provide a user friendly and efficient method for browsing the contents of a remote SPARQL endpoint. To speedup the efficiency of browsing we show how we can exploit the new features of HTML5, specifically its local storage, for realizing a dedicated caching mechanism. We discuss the various caching approaches that could be used and we propose a mechanism for the problem at hand. The experimental evaluation has shown that the proposed cache can speedup the browsing experience by 73%, regardless of the size of the contents of the remote endpoint, offering a smooth and fast browsing of any SPARQL endpoint. Finally, we present a client-side SPARQL endpoint browser that has been developed which supports the proposed caching mechanism, is based on client-side technologies and follows the principles of responsive web design.

Μηχανισμοί Προσωρινής Μνήμης για Πλοήγηση Σημείων Σύνδεσης SPARQL

Περίληψη

Ολοένα και περισσότερη δομημένη πληροφορία δημοσιεύεται σύμφωνα με τις αρχές των διασυνδεμένων δεδομένων (Linked Data), η οποία είναι διαθέσιμη μέσω σημείων σύνδεσης SPARQL (SPARQL endpoints). Συχνά δεν προσφέρεται η δυνατότητα περιήγησης αυτής της δομημένης πληροφορίας μέσω ιστοπεριηγητών (Web browsers). Αυτό σε συνάρτηση με την αυξανόμενη χρήση έξυπνων συσκευών (κινητά τηλέφωνα, tablets, κτλ) που έχουν ιστοπεριηγητές, οδήγησε στην εκπόνηση αυτής της μεταπτυχιακής εργασίας της οποίας στόχος είναι η δημιουργία μίας φιλικής και αποδοτικής μεθόδου με την οποία ένας χρήστης θα μπορεί να περιηγηθεί στα περιεχόμενα ενός απομακρυσμένου σημείου σύνδεσης SPARQL. Για να επιταχύνουμε την περιήγηση αναλύουμε τρόπους με τους οποίους μπορούμε να εκμεταλλευτούμε τις νέες δυνατότητες που μας δίνει η HTML5 και συγκεκριμένα αυτό της τοπικής αποθήκευσης (local storage) έτσι ώστε να μπορέσουμε να δημιουργήσουμε ένα μηχανισμός προσωρινής αποθήκευσης (cache). Προτείνουμε διάφορες εναλλακτικές για την προσωρινή αποθήκευση και στην συνέχεια αναλύουμε και αξιολογούμε πειραματικά τον προτεινόμενο μηχανισμό προσωρινής αποθήκευσης. Η πειραματική αξιολόγηση έδειξε ότι με την χρήση της προσωρινής μνήμης η εμπειρία περιήγησης βελτιώνεται κατά 73%, ανεξάρτητα από το μέγεθος των περιεχομένων της απομακρυσμένης πηγής, προσφέροντας μία ομαλή και γρήγορη περιήγηση, για οποιαδήποτε απομακρυσμένο σημείο σύνδεσης SPARQL. Τέλος, αναπτύχτηκε ένας περιηγητής για σημεία σύνδεσης SPARQL που βασίζεται μόνο σε τεχνολογίες πελάτη, υποστηρίζει τον προτεινόμενο μηχανισμό προσωρινής αποθήκευσης και είναι σύμφωνος με τις αρχές σχεδίασης για κινητές και σταθερές συσκευές.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Γιάννη Τζίτζικα, για την εμπιστοσύνη που μου έδειξε, αναθέτοντάς μου τη συγκεκριμένη διπλωματική εργασία και για την ορθή καθοδήγηση και ουσιαστική συμβολή του στην εκπόνηση της. Ακόμη θα ήθελα να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στην κ. Ειρήνη Φουντουλάκη για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή.

Επίσης ευχαριστώ το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την υποτροφία που μου προσέφερε, καθώς και για την πολύτιμη υποστήριξη σε υλικοτεχνική υποδομή και τεχνογνωσία.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαιτέρως τους γονείς μου, για την συμπαράσταση και την υποστήριξη που μου προσέφεραν καθ' όλη την διάρκεια των σπουδών μου.

στους γονείς μου

Contents

1 Introduction **3**
- 1.1 Motivation . . . 4
- 1.2 Goals and Approach . . . 4
- 1.3 Thesis Overview . . . 6

2 Background and Related Work **7**
- 2.1 Background . . . 7
  - 2.1.1 Semantic Web . . . 7
  - 2.1.2 SPARQL . . . 8
  - 2.1.3 Hybrid applications . . . 9
  - 2.1.4 Client side databases . . . 10
- 2.2 Related Work . . . 14
  - 2.2.1 Server-provided and Client-side Browsing of SPARQL endpoints . . . 14
  - 2.2.2 Caching . . . 15
  - 2.2.3 Client-provided and Browsing of SPARQL endpoints . . . 15
  - 2.2.4 Other related approaches . . . 16
  - 2.2.5 Our placement . . . 16

3 The Client-side Browsing of SPARQL Endpoints **17**
- 3.1 Interaction Model . . . 17
- 3.2 General principles . . . 25
- 3.3 SPARQL Endpoint Browsing Example . . . 25
- 3.4 Screenshots . . . 29

4 Caching Approaches **37**
- 4.1 Approaches . . . 37
- 4.2 Cache Refresh . . . 40
- 4.3 The adopted Caching Mechanism . . . 40

5 Implementation and Application **43**
5.1 Used Libraries and Applicability 43
5.2 Cache Implementation 44
5.2.1 Pilot Phase 44
5.2.2 Final Phase 48
5.3 Difficulties that we Encountered 49
5.4 How to use 50

6 Experimental Evaluation of the Cache Performance **53**
6.1 Measures 53
6.2 Metrics 54
6.3 Used SPARQL endpoints 54
6.4 Series of Requests 55
6.4.1 System Initialization Queries 56
6.4.1.1 SPARQL version of remote endpoint 56
6.4.1.2 Classes - SPARQL 1.0 56
6.4.1.3 Classes - SPARQL 1.1 57
6.4.1.4 Properties - SPARQL 1.0 58
6.4.1.5 Properties - SPARQL 1.1 59
6.4.1.6 Individuals - SPARQL 1.0 60
6.4.1.7 Individuals - SPARQL 1.1 61
6.4.1.8 Label and Description 62
6.4.2 Card Generation Queries 62
6.4.2.1 Class 62
6.4.2.2 Property 63
6.4.2.3 Individual 64
6.4.2.4 Incoming properties 64
6.4.2.5 Outgoing properties 64
6.4.2.6 Instances of Incoming properties 65
6.4.2.7 Instances of Outgoing properties 65
6.5 Carried out Experiments 66
6.6 Synopsis of the Experimental Results 71

7 Discussion **73**
7.1 Querying 73

8 Concluding Remarks and Future work **79**

A Appendix **81**
A.1 Keyword Search Queries 81
A.2 ASK Queries 88

A.3 Count Resources Queries 94
A.4 Labels and abstracts 100

List of Figures

2.1 Semantic Web basic layers 8
2.2 SPARQL Query Language for RDF 9
2.3 Native, HTML5 and Hybrid Development 10

3.1 Application Home Screen 18
3.2 Class Card representation 21
3.3 Property Card representation 23
3.4 Individual Card representation 24
3.5 ToyExample example graph 25
3.6 Home Screen 29
3.7 Settings Screen 30
3.8 Language configuration Screen 30
3.9 Advanced settings Screen 31
3.10 System initialization - Classes tab 31
3.11 System initialization - Properties tab 32
3.12 System initialization - Individuals tab 32
3.13 Settings expanded panel Screen 33
3.14 System statistics Screen 33
3.15 Class card Screen - Student 34
3.16 Individual card Screen - Yannis 34
3.17 Individual card Screen - UoC 34
3.18 Individual card Screen - Crete - Incoming properties 35
3.19 Individual card Screen - Crete - Outgoing properties 35
3.20 Individual card Screen - Vaggelis 35
3.21 Property card Screen - lives 36

5.1 Client-side SPARQL browser implementation 47
5.2 Database diagram 50
5.3 PascoLink on Nexus6x 51
5.4 PascoLink on iPhone6 52

6.1 (b)-caching approach - Initialization time 66

6.2 (b)-caching approach - Cache selection time. 67
6.3 (b)-caching approach - Cache insertion time. 67
6.4 (c)-caching approach - Card creation time. 68

7.1 Keyword searching on subjects, predicates and objects 76
7.2 Keyword searching on classes, properties and individuals . . . 76

List of Tables

2.1 Database features of local storing 12
2.2 Client side databases applicability 13

4.1 Advantages and disadvantages of caching approaches 39
4.2 Refreshing policy 40

5.1 Applicability on desktop browsers 44
5.2 Applicability on mobile browsers 44

6.1 Initialization Speedup for (b)-cache 69
6.2 Cache Card Generation Speedup for (c)-cache 70
6.3 Overall Cache Speedup 70

List of Algorithms

1 The (c)-caching approach algorithm for Card Generation . . . 41
2 The (b)-caching approach algorithm for System Initialization . 42

Chapter 1

Introduction

The World Wide Web (www) has radically changed the world by connecting different places and introducing a new era of sharing knowledge and information. However, most users could only navigate to websites and not contribute to their content. The information was static and could be updated only by experts. The second wave of Web (Web 2.0) provided the user with the capability of interaction and collaboration among other, creating dynamic web content with user-centered information. This was closer to the original vision of Tim Berners-Lee of a "collaborative medium, a place where we could all meet, read and write" [1].

Nowadays, it is easy to create web pages and publish them to the web. Now anyone, anywhere can access them using a Web Browser and make use of their content. Nevertheless, this information can be exploited only by people rather than machines. Users can understand the content of websites and navigate through links. In contrast with humans, machines cannot process data and use the available information with a meaningful purpose.

However, it is very difficult to develop software that collects information from the web to perform a specific function, for example organize a business trip according to specific preferences. This problem is a major challenge in research for more than a decade. If the available information was in a form comprehensible to machines then tools could take advantage of it and use it. Regarding this perspective, data from different sources could be connected together and it would be possible to create new knowledge derived from a combination heterogeneous data sources. This is the main purpose and vision of the Semantic Web.

The Semantic Web is a Web of Data and the goal is to allow the vast range of web-accessible information and services to be more effectively exploited by both human and automated tools. To facilitate this process, RDF and OWL have been developed as standards formats for the sharing and integration of

data and knowledge. These data and knowledge constitute a form of rich conceptual schemas called ontologies. These languages and the tools developed to support them, have rapidly become standards for ontology development and deployment. They exhibit an incremental use, not only in research labs, but also in large scale IT projects. Relational databases or XML need specific query languages (SQL, Xquery, etc.), the Web of Data typically represented using RDF. The RDF data can be represented as triples and stored in a specialized database, called TripleStore (SPARQL endpoint, warehouse). This data format, needs its own, RDF-specific query language and facilities. This is provided by the SPARQL query language and the accompanying protocols. SPARQL makes it possible to send queries and receive results, e.g., through HTTP or SOAP.

The SPARQL language is a language that can query remote Triplestores. This query language in combination with the latest web technologies as Ajax, Json and Http request can provide a very useful and fast tool in order to browse the contents of a remote SPARQL Endpoint.

1.1 Motivation

There is already a plethora of SPARQL endpoints and their number keeps increasing. The amount of data to be managed is stretching the scalability limitations of SPARQL endpoints that are conventionally used to manage Semantic Web data. At the same time, the Semantic Web is increasingly reaching end users who need efficient and effective browsing of the contents of these queryable datasets and this is the reason why browsable HTML pages are also provided (by the owner of an endpoint) in many cases. However, not every SPARQL endpoint provides this facility. On the other side, a significant percentage of users now have and use smart devices (phones, tablets) all having the ability to connect to the internet and therefore they are all equipped with internet browsers (and therefore with JavaScript). Therefore it is worth investigating whether a user with his/her web browser could survey the contents of a SPARQL endpoint even if the server side does not provide any browsable page.

1.2 Goals and Approach

The purpose of this work is to elaborate on how one can use his/her internet browser to scan through the contents of a remote SPARQL endpoint. The primal object of this thesis is the creation of a client side SPARQL browser.

To reach this objective, in this thesis, we investigate a client-side approach, i.e. an approach that requires having only a web browser. Note that a client side approach is directly applicable over any SPARQL endpoint and it does not require any deployment or operational maintenance. Furthermore, this approach is also open in the sense that the client could even himself change the code, altering the way he wants to navigate through the remote SPARQL endpoint.

To maximize the utilization of the client's resources thus increasing the efficiency of browsing, we present how we can exploit the new features that HTML5 offers (local storage), providing a caching mechanism. We discuss the various approaches that could be used for caching and then we present a sophisticated caching mechanism for the problem at hand.

The experimental evaluation has shown that the FreqData caching approach speeds up the system's initialization time approximately by 99% and that the URI-based caching approach speeds up the generation of a detailed resource card by 46% on average. Generally, taking into account the aforementioned speedup that FreqData and URI-based caching approaches offers on system's initialization time and card generation time. We can conclude that the cache speeds up the browsing experience approximately by 73% on average, offering a smooth and fast browsing of any SPARQL endpoint without the creation of any server side implementation.

Also note that client-side caching is also beneficial for the server side (i.e. the SE) in the sense that it reduces the load of the SE. In a nutshell, the key contributions of this work are:

- It is the first work on "client-side only" caching for browsing remote SPARQL endpoints.
- We discuss various caching approaches that could be used for this problem, their pros and cons, and then we propose a dedicated caching mechanism.
- We present the results of a comparative experimental evaluation according to various perspectives:
  (a) Caching methods
  (b) Cache size
  (c) Client side databases
  (d) SE

1.3 Thesis Overview

The body of this work is organized in seven main chapters. The next chapter provides the relevant background and describes the main prerequisites. The following chapter provides the client-side browsing approach with emphasis on the caching mechanism. Then the next chapters describe the implementation, the experimental analysis, discussion about search and finally concludes with future work that's worth researching. The thesis is organized as follows:

Chapter 2 provides the background in order to implement a client-side SPARQL browser with caching, then presents some relative approaches involving caching.

Chapter 3 presents the client-side browsing approach with emphasis on the caching mechanism. Specifically, it describes the interaction model, then discusses various possible caching mechanisms, then presents the proposed caching mechanism and finally defines and analyses cache refresh policies.

Chapter 4 provides implementation and applicability details, then describes how the cache mechanism was implemented and then provides indicative screen-shots.

Chapter 5 focuses on cache performance and reports detailed experimental results. Also analyses the measures and metrics that are used in order to export performance results over the cache.

Chapter 6 provides a discussion about how the cache could be exploited by keyword search, as well as the increase in overall efficiency as experienced by the user.

Chapter 7 chapter draws some conclusions about this work and identifies issues for further work and research.

Finally, Appendix A gives an overview of some SPARQL queries that are used in order to create the client side SPARQL browser.

Chapter 2

Background and Related Work

2.1 Background

In order to fulfil the objective of a client-side browsing of SPARQL endpoints with caching we have to exploit technologies like HTML5 [2], jQuery, jQuery mobile framework, RDF/RDFS [3] and SPARQL [4].

The HTML5 is the next generation of HTML that provides new features that are necessary for modern web applications. Feature like Web Sql Database [5] that is capable of storing data locally. RDF/RDFS [3] is a standard format language for the sharing and integration of data and knowledge of rich conceptual schemas called ontologies. SPARQL is a semantic query language capable of retrieving and manipulating data stored in RDF format. JQuery/mobile framework that is a web library that simplifies writing JavaScript with combination of versatility, responsiveness and extensibility that modern desktop and mobile browsers require.

2.1.1 Semantic Web

The Semantic Web provides a common framework that allows data to be shared and reused across applications. Its components are deployed in the layers of Web technologies and specifications as represented in Figure 2.1.

There are five main components of the Semantic Web:

1. URI - Uniform Resource Identifier: is a format for web identifiers that is widely used on the World Wide Web. The Semantic Web uses URIs to represent most kinds of data.

2. RDF - Resource Description Framework [6, 7] is used by Semantic Web to describe data uniformly, allowing it to be shared. It is a general metadata format used to represent information about Internet resources. It

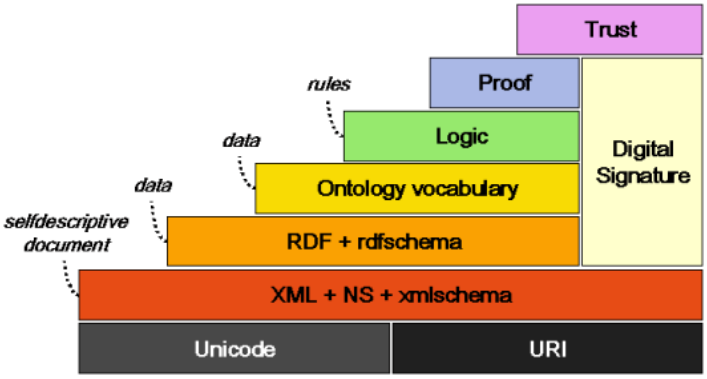

Figure 2.1: Semantic Web basic layers

extends the expressive capability of Web augmenting human-readable web pages with machine-processable information.

3. RDF Schema is a language used by the Semantic Web to describe the data properties used in RDF. It provides mechanisms for describing resources and relationships between these resources. Also enables the ability to reason over RDF data enabling the discovery of implicit data from explicitly represented ones. The RDFS vocabulary descriptions are also RDF [8].

4. Ontologies are used to represent the structure of knowledge domain [29]. The Semantic Web uses OWL, Web Ontology Language. Applications need that language in order to process data rather than just display it. OWL adds the possibility of reasoning to data by identifying and describing relationships between data items. Ontologies are defined independently from the actual data and reflect a common understanding of the semantics of the domain. It provides definitions of classes, relations, functions, constraints and other objects.

5. Logic - Inference is useful to derive new data from data. A common example is the property transitivity. If an element has a type A and the type A is a subtype of type B then the element has also the type B.

2.1.2 SPARQL

A data management system requires a language to query the data it contains. For this purpose, the Semantic Web group at W3C has published the SPARQL Protocol and RDF Query Language (SPARQL) recommendations.

The main aspect of these recommendations was to support a query language. Although some clauses such as SELECT and WHERE may look like the popular Structured Query Language (SQL) or RDBMS, SPARQL is based on the notion of triples and not on relations.

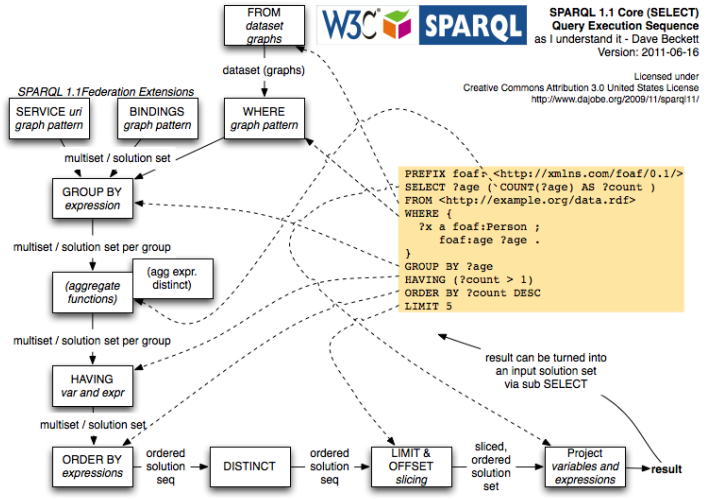

Figure 2.2: SPARQL Query Language for RDF

Answering a SPARQL query implies some pattern-matching mechanisms between the triples of the query and the data set. This requires research in the field of query processing and optimization that goes beyond the state of the art of the relational model.

2.1.3 Hybrid applications

Applications can generally be broken down into native, hybrid and web apps. Going the native route allows to use all the capabilities of a device and operation system, with the minimum of performance overhead on a given platform.

However, building a web app allows code to be ported across platforms, which can dramatically reduce development time and cost. Hybrid apps combine the best of both approaches, using a common code base to deploy native-like apps to a wide range of platforms. There are two approaches to build a hybrid app:

1. WebView app
   The HTML, CSS and JavaScript code base runs in an internal browser (called WebView) that is wrapped in a native app. Some native APIs are exposed to JavaScript through this wrapper.

2. Compiled hybrid app
   The code is written in one language (such as C# or JavaScript) and gets compiled to native code for each supported platform. The result is a native app for each platform, but less freedom during development.

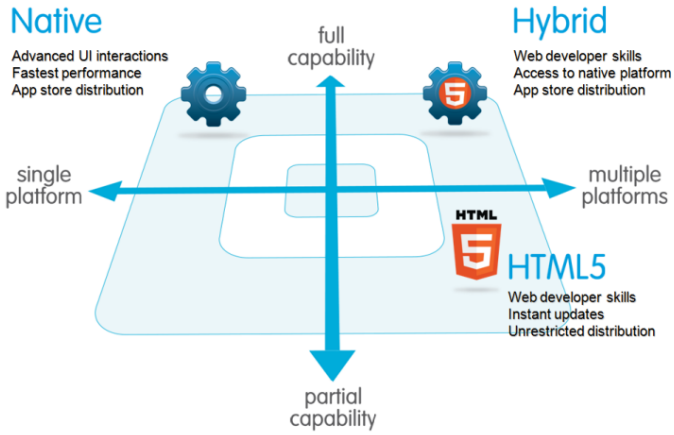

Figure 2.3: Native, HTML5 and Hybrid Development

We used the first approach to create the client side SPARQL browser. The reason that we selected the first approach was that gave us the opportunity to easily build the application for every software of mobile or desktop system using web technologies.

Hybrid apps as WebView apps are like any other apps exist on desktop and mobile platforms. Also can be installed on every device and can be found in app stores. Like the websites on the internet, hybrid apps are build with a combination of web technologies like HTML, CSS and Javascript. The key difference is that hybrid apps are hosted inside a native application that utilizes a platform's WebView (you can think WebView as a chromeless window of device's integrated browser that's typically configured to run fullscreen). This can enable to access mobile hardware capabilities such as the accelerometer, camera, contacts etc.

2.1.4 Client side databases

The modern browsers with support of HTML5 include new features of offline storing capabilities that comes with a fully functional API. Each of these storing capabilities serves different purposes, and therefore has a different approach and specific advantages and shortcomings. Nevertheless, the common objective of all is to overcome the limitations of legacy client-side storing

mechanisms [9] and reducing load on web servers. Below we list the client side databases approaches:

1. Web storage [10]
   The database allows to store key-value pair data directly on the client side in the browser for repeated access across requests or to be retrieved long after you completely close the browser (also referred to as DOM Storage, HTML5 Storage, Local Storage, Offline Storage).

2. Indexed Database [11]
   The database is also a client side storage mechanism for storing data based on non-relational database known as NoSQL databases with support for transactions and indexing.

3. Web SQL Database [5]
   In this case, the application allows access to a relational database known as SQLite through an asynchronous JavaScript interface that shows off the power of databases in the browser. This database support transactions and SQL querying.

The differences of each one local storing approaches is described in the following table.

Table 2.1: Database features of local storing

Features	**Web Storage-Session storage**	**Web Storage - Local Storage**	**Web SQL Database**	**Indexed Database**
Data types	String only, as key-value pairs	String only, as key-value pairs	SQL data types	Object Store with objects and keys
Default Max Size	5 MB per origin (Can be increased upon user verification)	Limited only by system memory	5 MB per origin (Can be increased upon user verification)	5 MB per origin (Size can change between different browsers)
Persistence	On disk until deleted by user (delete cache) or by the app	Survives only as long as its originating window or tab	On disk until deleted by user (delete cache) or by the app	On disk until deleted by user (delete cache) or by the app
Availability	Shared across every window and tab of one browser running same web app	Accessible only within the window or tab that created it	Shared across every window and tab of one browser running same web app	Shared across every window and tab of one browser running same web app
Synchronization	Synchronous API	Synchronous API	Asynchronous API	Asynchronous API
Query	Through serialization /deserialization of string object	Through serialization /deserialization of string objects	SQL Query	Cursor APIs, Key Range APIs and Application Code
Standardization	W3C Candidate Recommendation 09 June 2015	W3C Candidate Recommendation 09 June 2015	W3C Working Group Note 18 November 2010 (deprecated)	W3C Recommendation 08 January 2015

The browser support of each of the client side database of the most well known, modern browsers is described in the following table.

Table 2.2: Client side databases applicability

Browsers	Web Storage	Indexed Database	Web SQL Database
IE	8.0+	10+ (Partial support)	-
Edge	12.0+	12+ (Partial support)	-
Firefox	3.5+	4.0+	-
Chrome	4+	11+	4.0+
Safari	4.0+	7.1+ (Buggy behavior)	3.1+
Opera	11.5+	15+	11.5+
iOS Safari	3.2+	8+ (Buggy behavior)	3.2+
Opera Mini	-	-	-
Opera Mobile	12+	30	12+
Android Browser	2.1+	4.4+	2.1+
Chrome for Android	44	44	44
Firefox for Android	40+	40	-
IE Mobile	10+	10+ (Partial support)	-
Blackberry Browser	7.0+	10	7+
UC Browser for Android	9.9	-	9.9

2.2 Related Work

At first (chapter 2.2.1) we discuss server-side approaches, then (section 2.2.2) we present some relative approaches involve caching. Afterwards (chapter 2.2.3) we discuss approaches that involve both the server and the client, other relative approaches (chapter 2.2.4), and finally (section 2.2.5) we describe the placement of the current work.

2.2.1 Server-provided and Client-side Browsing of SPARQL endpoints

Usually server-side tools produce HTML pages that the user can browse using a browser. Some indicative systems are listed next. Tabulator [12] (a generic data browser which provides ways of browsing RDF resources on Web), Disco - Hyperdata Browser [13] (a browser that handles the Semantic Web as an unbounded set of resources), Swoogle [14] (a specialized web based data browser used for discovering, analyzing and indexing of data from datasets published on the Web with Semantic Web technologies), Longwell[1] (a web-based faceted browser, considered as a combination of the flexibility of the RDF data model and the effectiveness of the faceted browsing paradigm), Virtuoso Faceted Browser[2] (a keyword-based faceted browser) that support changing the focus from one set of resources to a related one (known as pivoting).

Moreover there are several SPARQL clients, i.e. clients that help the users to formulate SPARQL queries. For instance, YASGUI [15] lists and categorizes (according to their functionality) currently existing SPARQL endpoints. YASGUI is a web-based SPARQL client that functions as a wrapper for both remote and local endpoints. It integrates linked data services and web APIs to offer features such as auto-completion and endpoint lookup and browsing. YASGUI is built using SmartGWT toolkit[3], jQuery, and uses new HTML5 functionalities such as local storage and client-side generation of files.

NL-Graphs [16] is another system that falls in this category that provides a hybrid query solution that includes graph-based and natural language querying.

[1] http://simile-widgets.org/
[2] http://lod.openlinksw.com
[3] http://www.smartclient.com/product/smartgwt.jsp

2.2.2 Caching

The term caching refers to techniques that are used in several systems of various levels (from processors to web-search engines [17]) to reduce processing costs and attain faster response times. There are caching techniques that focuses on SPARQL query and answer process [18]. As regards caching there are works, like [19] that focuses on caching for the server side, i.e. such caches that contain the results of SPARQL queries.

Server-side caching is also the subject of [20] that focuses on improving the performance of triple stores by caching SPARQL queries results.

2.2.3 Client-provided and Browsing of SPARQL endpoints

SPARKLIS [21] is a client-side expressive query builder for exploring SPARQL endpoints with guidance in natural language. It use query-based faceted search (it use YASGUI engine) as a way to reconcile the expressivity of formal languages and the usability of faceted search. Although that system can explore a SPARQL endpoint with natural language, it doesn't support cache speedup and browsing from one set of resources to a related one (pivoting).

Facet Browser [22] presents a facet browser for SPARQL endpoints, based on HTML5. Although it is a server-side solution it also tries to exploit some features of the HTML5 in order to exploit the resources of the client machine. In general, that system allows users to search and retrieve RDF triples based on a keyword, from public SPARQL endpoints. By using HTML5 Web Storage, the triples from the results can be saved in the browser, locally, for future use. The [22] also provides management functionalities over the stored data - capabilities to update, refresh, modify, delete and download the triples in various RDF formats: JSON-LD, Turtle, NTriples, RDF/XML, JSON, CSV. As regards local storage, HTML5 offers two different types of storage: local and session. They use the former because it stores data with no expiration date, which means the data will not be deleted and will persist when the user closes the browser and his session ends. The application allows the users to define details about the RDF triples before saving them in Web Storage. As regards the local storage, they store information in "records" with the following structure: mnemonic name, the graph name, set of triples. They use the JavaScript wrapper library `triplestore.js`. Although that system exploits the client side, the provided functionality cannot be considered as a cache in the sense that the user should explicitly request saving, refreshing or deletion of a locally stored dataset.

2.2.4 Other related approaches

In [23] the authors present a pre-fetching server-side approach, i.e. how instead of sending only one query, compute and sent an augmented query aiming at retrieving data that is potentially interesting for subsequent requests in advance.

2.2.5 Our placement

To the best of our knowledge, this is the first work that focuses on a pure client-side solution for providing browsing of SPARQL endpoints and pays special attention to client caching by presenting various caching approaches, propose a caching mechanism and finally experimentally evaluate the proposed caching mechanism. Probably the more related (so far) work is the client-part of [22], but as explained in the previous section it cannot be considered as a cache.

Chapter 3

The Client-side Browsing of SPARQL Endpoints

Here we describe our approach. At first we describe the interaction model (3.1), then we report some general principles of the system (3.2), then (3.3) we analyse and report a working example over the implemented client-side SPARQL browser and finally (chapter 3.4) we provide some screenshots of the application.

3.1 Interaction Model

We created an online platform that hosts all the functions that a user has the ability to execute as follows (Fig. 3.1):

1. The user can use the client side SPARQL browser as an online platform (Online Browser).

2. The user can run online custom benchmarks on a user specific rdf Triplestore and set the number of synchronization resources (Bench-Mark). By the term synchronization resources, we mean all the resources necessary for the query production, caching and user response.

3. We present all these queries that were used in order to create the client side SPARQL browser (Queries).

4. The user can choose and download the client side browser for desktop and mobile platforms (Download).

5. The user can be informed about the project through the presentation contained on the online platform (Presentation)

6. The user can contact the authors and developers of the client side SPARQL browser (Contact).

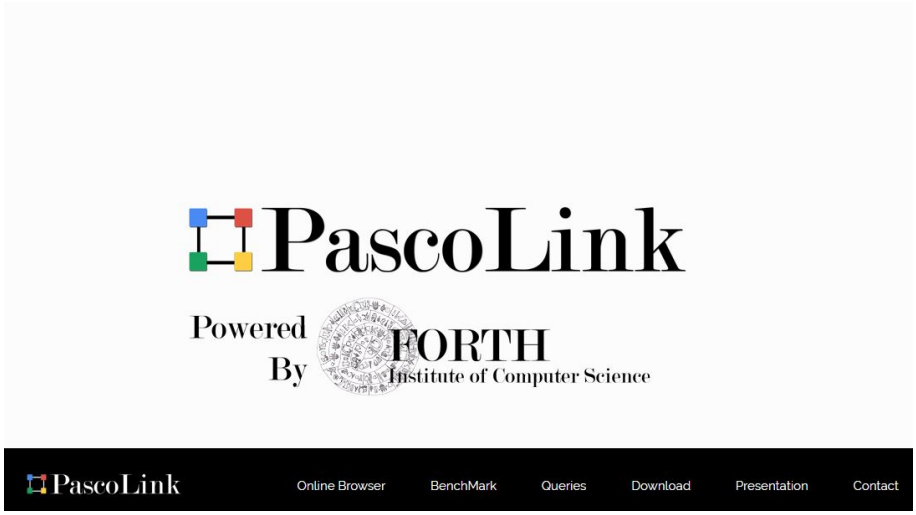

Figure 3.1: Application Home Screen

The process of the client side SPARQL browser is the following:
At the beginning the user to configure the system (**"Settings" button**) with the appropriate setting in order to browse or search the contents of the remote endpoint. The configuration consist of normal and advanced settings.

1. Normal Settings (Fig. 3.7)
   (a) The user must set the URL address of the remote endpoint.
   (b) The user can select from a set of example warehouses .
   (c) The user can set the language of the resource's label from a predefined list or a large list by clicking the "More language" button (Fig. 3.8).
   (d) The user can clear the contents of the client side SPARQL browser cache.
2. Advanced Settings (Fig. 3.9)
   (a) By clicking the button "Select specific Graph" we present all graphs of the remote endpoint. The user can select one or multiple graphs from the selected Triplestore.

(b) The user can set an inference rule to the selected remote endpoint. He can choose an inference rule with the same name as a selected graph or search the inference rule on a remote endpoint.

(c) The user can select a number of fields (URI, Label, Abstract) on the system when performing keyword search on a browsing category.

After the configuration is completed the users are redirected to home page with two possible options. The first option is to search to remote endpoint by keyword searching and the second option is to browse the contents of the remote endpoint.

By selecting the **"Search"** (Fig. 3.6) the user will be redirected to a new page that displays a text field and a search button. The user can set a keyword to search upon the selected remote endpoint by clicking the button "Search". When the searching is completed we provide three lists of searching results:

1. A list of all the resources matching the specified keyword as a **Subject** and the total number of occupancies, when the user clicks the "Subject" tab.

2. A list with all the resources matching the specified keyword as a **Predicate** and the total number of occupancies, when the user clicks the "Predicate" tab.

3. A list with all the resources matching the specified keyword as an **Object** and the total number of occupancies, when the user clicks the "Object" tab.

4. The user can have more information about the total number of Subjects, Predicates and Objects that exist on selected remote endpoint.

5. In every tab (Subjects, Predicates, Objects) a "More" button is displayed. The user by clicking the button the system synchronizes with the next 100 entities of each category.

By selecting **"Browse"** (Fig. 3.6), the user will be redirected to a new page after the execution of the queries to the remote endpoint. Then the following options and results are displayed to the user (Fig. 3.2, 3.3, 3.4).

1. A list is displayed to the user with the first 100 Class resources, when the user clicks the "Classes" tab.

2. A list is displayed to the user with the first 100 Property resources, when the user clicks the "Property" tab.

3. A list is displayed to the user with the first 100 Individual resources, when the user clicks the "Individual" tab.

4. The user can have more information about the total number of Classes, Properties, Individuals and Blank nodes that exist on the selected remote endpoint on each tab.

5. In every tab (Classes, Properties, Individuals) a text field is displayed. The user can type any text in the text field and the cached entities filter it, matching the keywords. Except for filtering the user can search matchings of text on the remote SPARQL endpoint by clicking the "Remote Search" button.

6. In every tab (Classes, Properties, Individuals) a "More" button is displayed. By clicking the button the system synchronizes with the next 100 entities of each category.

7. The user can also acquire a more detailed information card by clicking the "information" button from an expanded panel from the button "Info/Settings". The more detailed view consists of the selected URL of the remote endpoint, the selected graphs, the selected inference rules, the version of SPARQL that the remote endpoint supports and the system statistics.

8. The system statistics consist of specific diagrams of the total number of Classes, Properties and Individuals compared to cached entities.

9. The user can change the fields that are already selected on the advanced settings of the system, used to perform the keyword search on a browsing category (Class, Property, Individual) by selecting the button "Info/Settings".

The user can select a resource from the aforementioned lists and then the **resource's card** appears. The card of the selected resource firstly displays static information that consist of the **label**, the **description** and the **URI** of the selected resource. Secondly, displays dynamic information that synchronizing by clicking that consist of Schema Information, Incoming Properties, Outgoing Properties and Instances of each selected resource. The content of the above dynamic information differentiate between Classes, Properties and Individuals cards.

The **Class resource** (Fig. 3.2, 3.10) card consist of "Schema Information", "Incoming Properties", "Outgoing Properties", "Direct Instances" and "All Instances" as buttons. The "Schema Information" button of class card, if clicked collapse and display the following buttons:

- Super Classes
- Subclasses
- Equivalent To
- Disjoint with

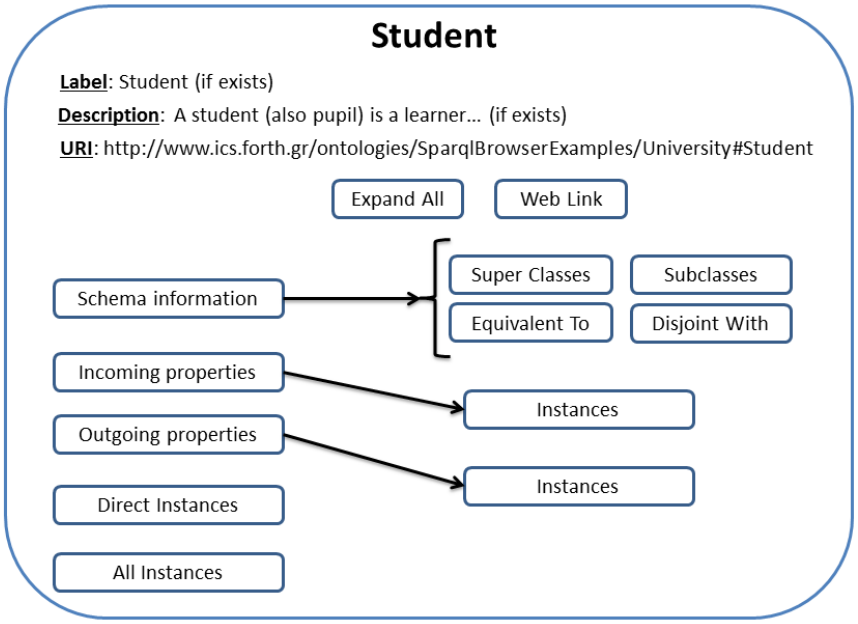

Figure 3.2: Class Card representation

When the above buttons are clicked they display the super classes, the subclasses, the equivalent classes and the disjoint classes of the selected resource as buttons, respectively. When the user clicks on these buttons he gets redirected to the clicked resource's card.

The "Incoming properties" and "Outgoing properties" buttons, when clicked enable the synchronization of the incoming or outgoing resources of the selected resource and display them as buttons. These buttons are separated into two clickable areas.

- By clicking on fist area (left, gray color) on each of incoming or outgoing resource button, it collapses and then the system synchronize and display the instances of the clicked incoming or outgoing property resource as buttons. By clicking on an instance button, the user redirects to the instance's card.
- By clicking on second area (right, black color) on each of incoming or outgoing resource button, the user redirects to the property's card.

The instances of card resource consist of the following categories and represented as buttons:

- Direct Instances
- All Instances

The "Direct Instances" and "All Instances" buttons, if clicked collapse and display instances as buttons. By clicking on an instance button, the user redirects to the instance's card. The direct instances consist of immediate connected instances of the selected resource. On the other hand, All Instances presupposes the existence of an inference rule that enables the instance reasoning on the remote endpoint. The instance reasoning enable the synchronization not only of the immediate connected instances, but all semantically associated connected resources of the selected resource.

The **Property resource** (Fig. 3.3, 3.11) card consist of "Schema Information", "Incoming Properties" and "Outgoing Properties". The "Schema Information" button of property card, if clicked collapse and display the following buttons:

- Super Properties
- Subproperties
- Domain
- Range

When the above buttons are clicked they display the super properties, the subproperty, the domain classes and the range classes of the selected resource as buttons, respectively. When the user clicks on these buttons he gets redirected to the clicked resource's card.

The "Incoming properties" and "Outgoing properties" buttons, when clicked enable the synchronization of the incoming or outgoing resources of the selected resource and display them as buttons. These buttons are separated into two clickable areas.

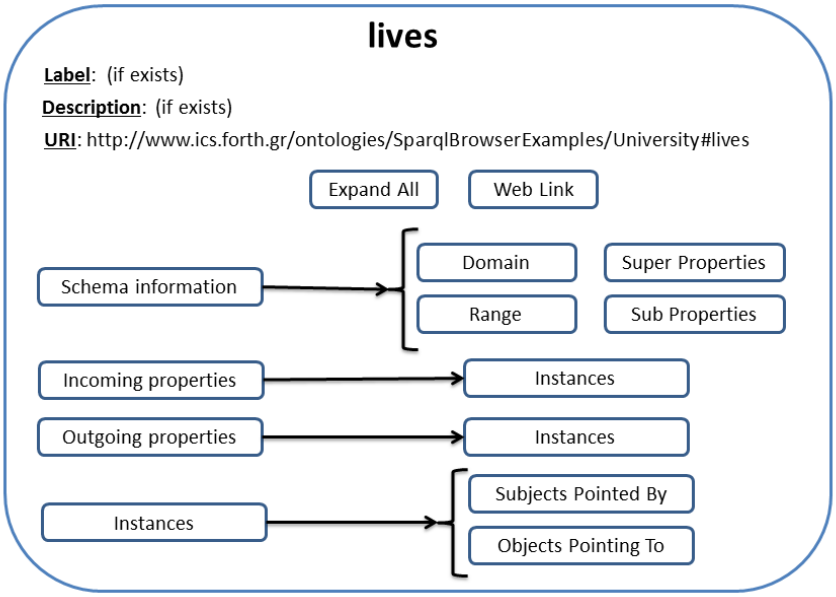

Figure 3.3: Property Card representation

- By clicking on fist area (left, gray color) on each of incoming or outgoing resource button, it collapses and then the system synchronize and display the instances of the clicked incoming or outgoing property resource as buttons. By clicking on an instance button, the user redirects to the instance's card.

- By clicking on second area (right, black color) on each of incoming or outgoing resource button, the user redirects to the property's card.

The instances of property resource consist of the following categories and represented as buttons:

- Subjects Pointed By

- Objects Pointing To

The "Subjects Pointed By" and "Objects Pointing To" buttons, if clicked collapse and display instances as buttons. The instances consist of entities that subject or objects to the selected resource. By clicking on an instance button, the user redirects to the instance's card

The **Individual resource** (Fig. 3.4, 3.12) card consists of Schema Information, "Incoming Properties" and "Outgoing Properties". The Schema Information of individual card consists of the following buttons:

- Type of
- Same As

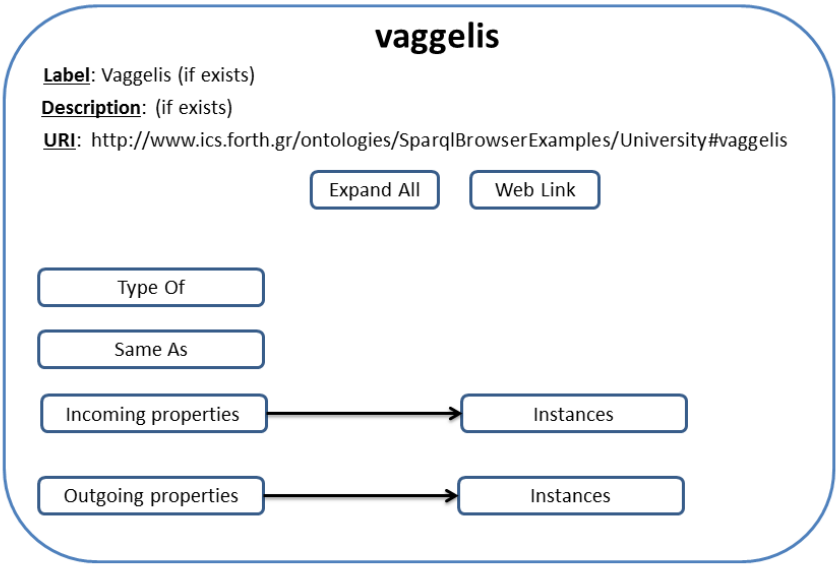

Figure 3.4: Individual Card representation

When the above buttons are clicked they display the class type and the same individuals of the selected resource as buttons, respectively. When the user clicks on these buttons he gets redirected to the clicked resource's card.

The "Incoming properties" and "Outgoing properties" buttons, when clicked enable the synchronization of the incoming or outgoing resources of the selected resource and display them as buttons. These buttons are separated into two clickable areas.

- By clicking on fist area (left, gray color) on each of incoming or outgoing resource button, it collapses and then the system synchronize and display the instances of the clicked incoming or outgoing property resource as buttons. By clicking on an instance button, the user redirects to the instance's card.
- By clicking on second area (right, black color) on each of incoming or outgoing resource button, the user redirects to the property's card.

In general the user is able **automate** the synchronization of the dynamic information of the selected resource (**"Expand all" button**) and head to the

corresponding HTML page of the recourse if exists (**"Web link" button**) (Fig. 3.10, 3.11, 3.12).

3.2 General principles

Whenever a list contains more than K elements then the system shows only the first K elements and a pagination button appears allowing the user to inspect the next chunk of K elements, and so on, until having consumed the entire list. The value of K that represents the pagination threshold is set to 25 elements per page to simplify the browsing in mobile and desktop browsers.

3.3 SPARQL Endpoint Browsing Example

In order to explain how the system works, we created an example graph named ToyExample (Fig 3.5) and describe some scenarios to show how the system automate the browsing of the following graph.

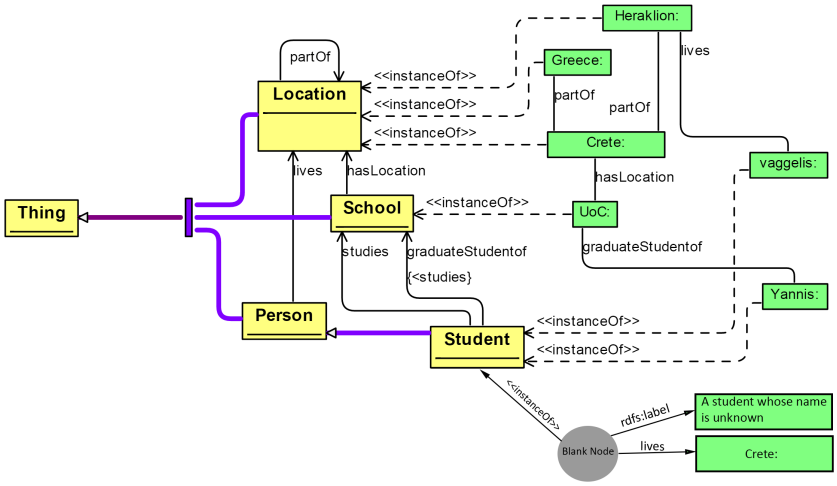

Figure 3.5: ToyExample example graph

This graph represents each entity's category or relationship between resources with different box color, line or specific symbols as follows:

- The Class resource is represented by yellow color box.
- The Property resource is represented by a line or an arrow. The name of the property is on the line or the arrow.

- The property is represented by a black line when connected and create a relationship between two individuals.
- The property is represented by an black arrow when connected and create a relationship between two classes. The class that the arrow pointing to is the range of the property otherwise is the domain of the property.
- The Individual resource is represented by green color box.
- The instances are represented by a black dot-arrow. The class that the dot-arrow pointing to is the type of the instance.
- The Blank nodes are represented by a gray circle.
- The subclass or superclass property is represented by a purple arrow among classes. The class that the arrow pointing to is the superclass of the class otherwise is the subclass of the class.
- The subproperty relationship between properties is represented by the "<" symbol, when there are two names of properties on a line.

The graph representations that we analysed above help us extract information about the classes, the properties, the individuals and the relationships among them. As a result, we observe that there are 4 classes, 5 properties, 6 individuals and 1 blank node as follows:

1. Classes
   (a) Location
   (b) School
   (c) Person
   (d) Student
      i. Subclass of Person
2. Properties
   (a) graduateStudentof
      i. Subproperty of studies
      ii. Domain: Student
      iii. Range: School
   (b) studies

i. Domain: Student
ii. Range: School

(c) hasLocation
i. Domain: School
ii. Range: Location

(d) lives
i. Domain: Person
ii. Range: Location

(e) partOf
i. Domain: Location
ii. Range: Location

3. Individuals

(a) Crete
i. Instance of Location
ii. partOf Greece

(b) Vaggelis
i. Instance of Student
ii. Lives at Heraklion

(c) Yannis
i. Instance of Student
ii. graduateStudentof UoC

(d) UoC
i. Instance of School
ii. hasLocation at Crete

(e) Greece
i. Instance of Location

(f) Heraklion
i. Instance of Location
ii. partOf Crete

4. Blank Node

(a) Instance of Student

(b) Label: "A student whose name is unknown"

(c) lives on Crete

The above representations of the example graph can define some facts as follows:

1. There are three students **Yannis**,**vaggelis** and **a student whose name is unknown (b2832297)**
2. Yannis is a graduate student of UoC
3. Uoc located in Crete
4. Crete is part of Greece
5. vaggelis lives in Heraklion
6. Heraklion is part of Crete
7. The student whose name is unknown lives in Crete

These previously defined facts can be answered by browsing the graph through the client side browser. Firstly the URL address of the remote endpoint have to be configured (Fig. 3.7) in order to enable to browse the contents of the defined graph.

After the configuration is finished the contents are browsable by clicking the "browse" option (Fig. 3.6)

The contents of the SPARQL endpoint are divided into three tabs that represent Class (Fig. 3.10), Property (Fig. 3.11) and Individual resources (Fig. 3.12).

In the Class tab (Fig. 3.10) we click on Student class resource to display the card of this resource and we expand the direct instances. We observe (Fig. 3.15) that there are three students **Yannis**, **vaggelis** and **a student whose name is unknown (b2832297)**.

From the Student class card we click on Yannis resource (Fig. 3.15) to display the card of this resource. We expand the outgoing properties and then the graduateStudentof property that has UoC as an instance. As a result, we observe (Fig. 3.16) that the **Yannis is a graduate student of UoC**.

From the Yannis individual card we click on UoC (Fig. 3.16) to display a detailed card of this resource. If we click on the expand all button we can observe (Fig. 3.17) the previously defined fact from the incoming properties and from the outgoing properties the fact that **UoC is located in Crete**.

From the Uoc individual card we click on Crete (Fig. 3.17) to display a detailed card of this resource. We expand the outgoing properties and then the partOf property that has Greece as an instance. As a result, we observe (Fig. 3.19) that the **Crete is part of Greece**.

From the Crete individual card (Fig. 3.18) we expand the incoming properties and then the lives property that has b2832297(a student whose name is unknown) as an instance. As a result, we observe (Fig. 3.18) that the **student whose name is unknown lives in Crete**.

In the Individual tab (Fig. 3.12) we click on vaggelis individual resource to display the card of this resource and we expand the outgoing properties and then the lives property. We observe (Fig. 3.20) that **vaggelis lives in Heraklion**.

3.4 Screenshots

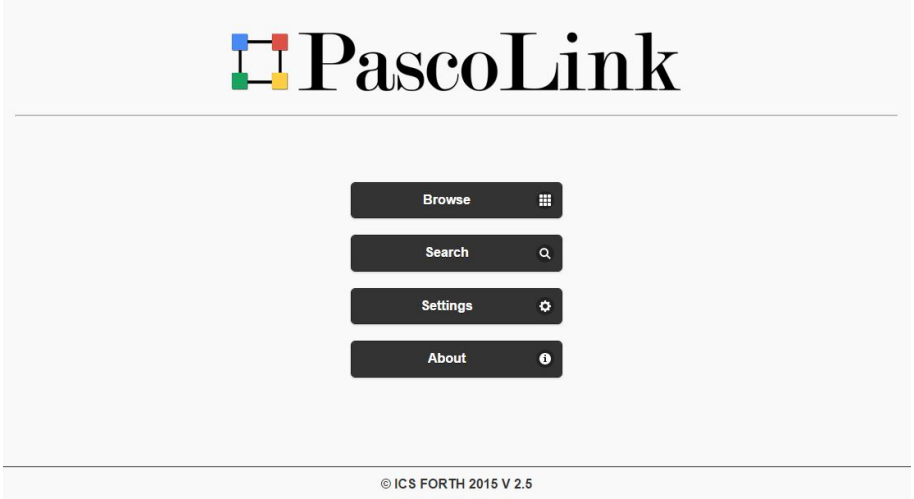

Figure 3.6: Home Screen

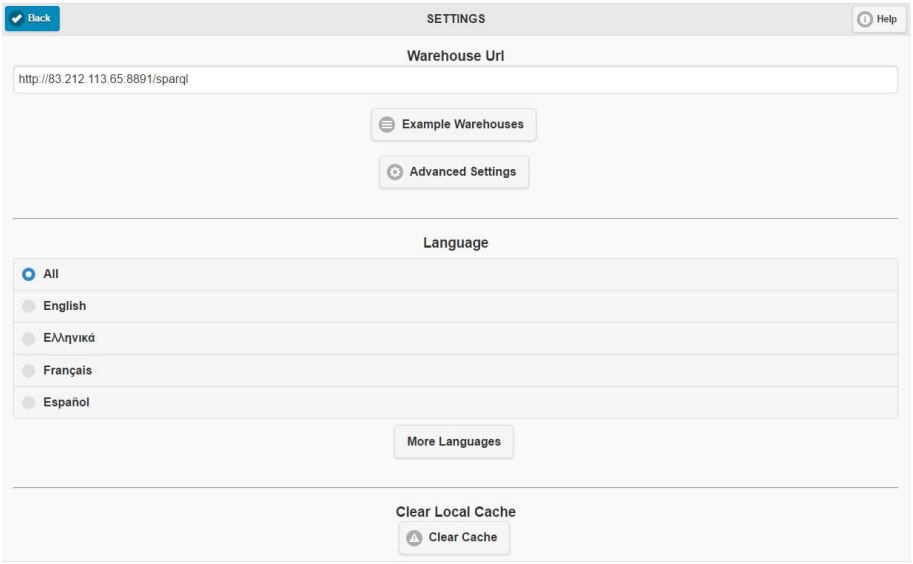

Figure 3.7: Settings Screen

Figure 3.8: Language configuration Screen

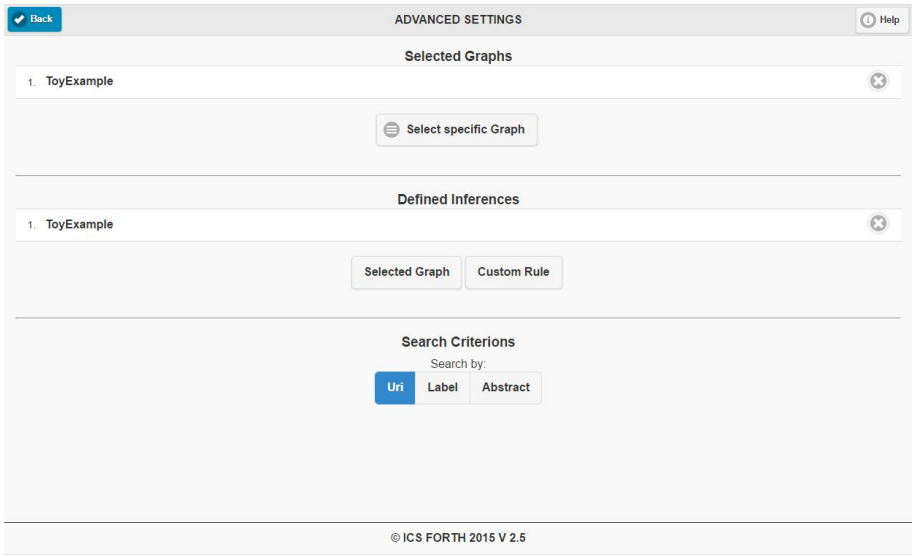

Figure 3.9: Advanced settings Screen

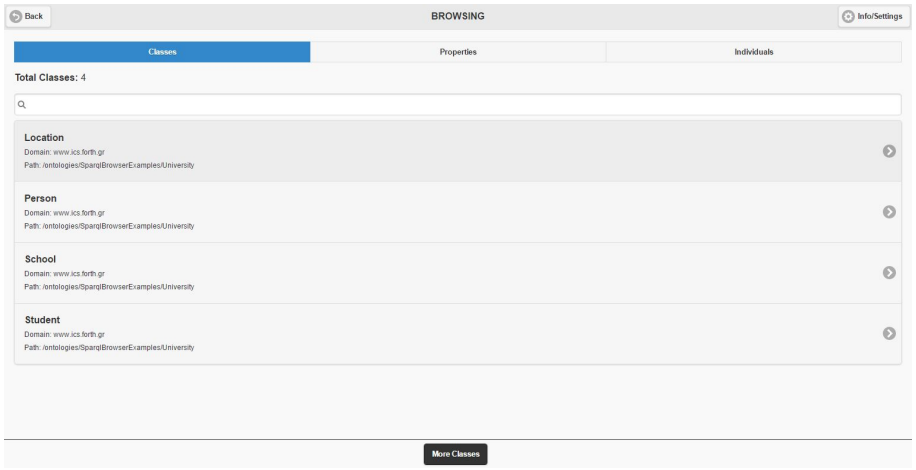

Figure 3.10: System initialization - Classes tab

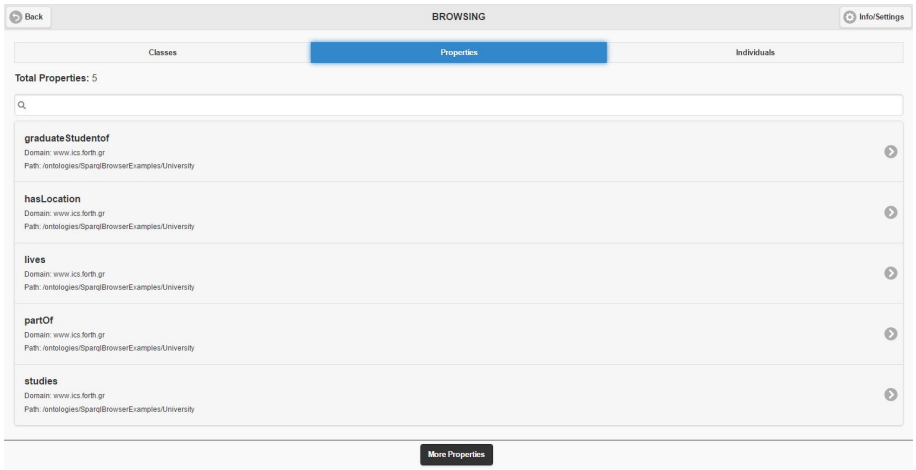

Figure 3.11: System initialization - Properties tab

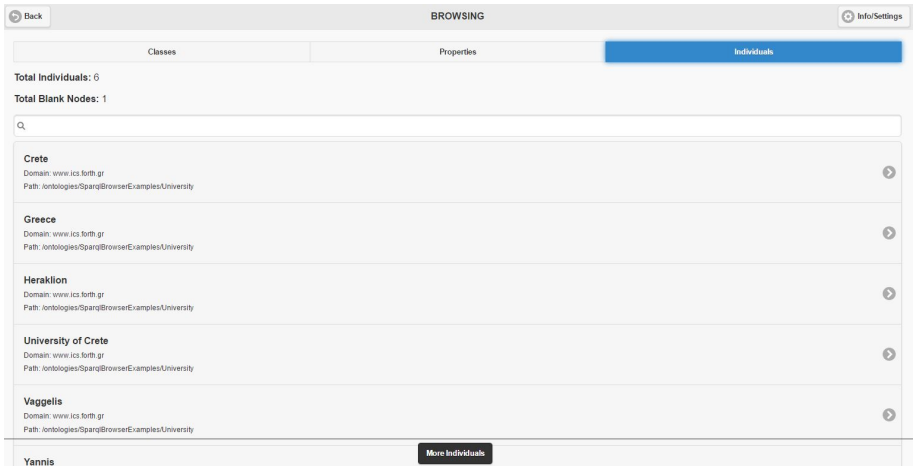

Figure 3.12: System initialization - Individuals tab

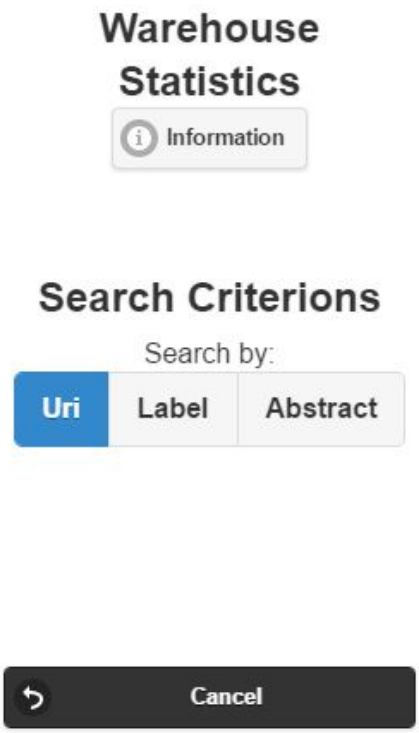

Figure 3.13: Settings expanded panel Screen

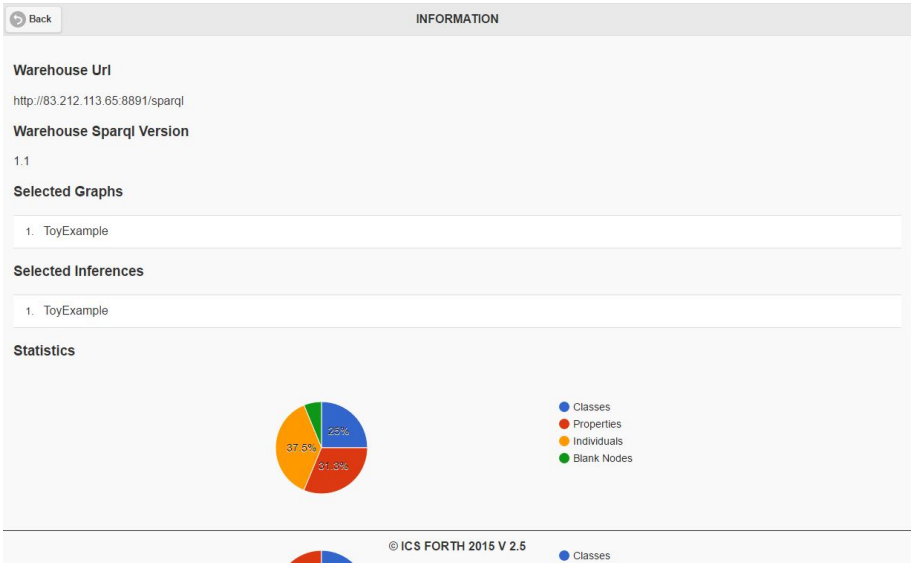

Figure 3.14: System statistics Screen

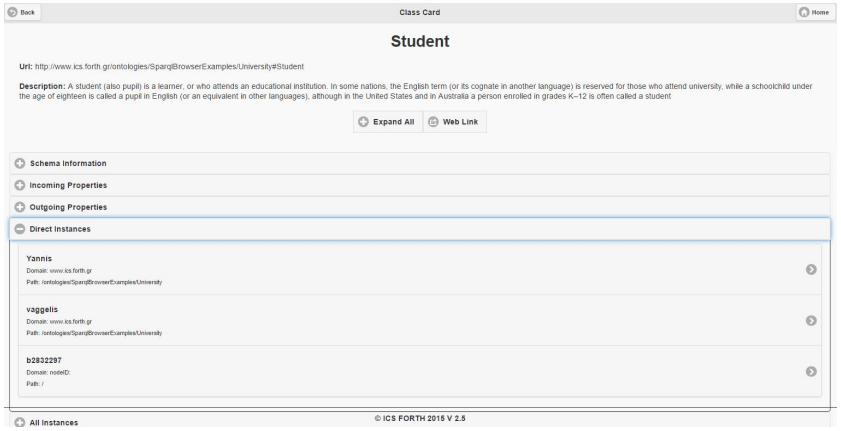

Figure 3.15: Class card Screen - Student

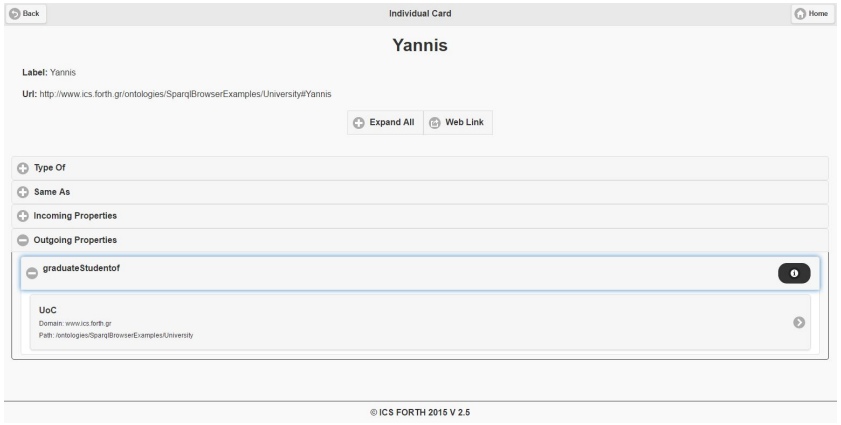

Figure 3.16: Individual card Screen - Yannis

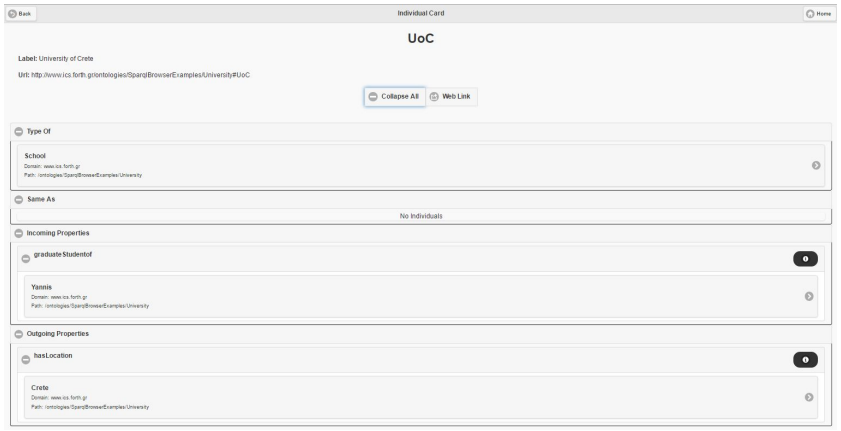

Figure 3.17: Individual card Screen - UoC

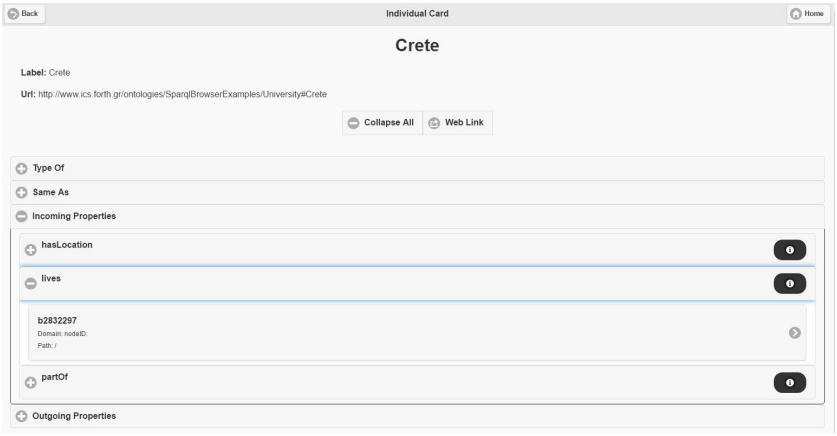

Figure 3.18: Individual card Screen - Crete - Incoming properties

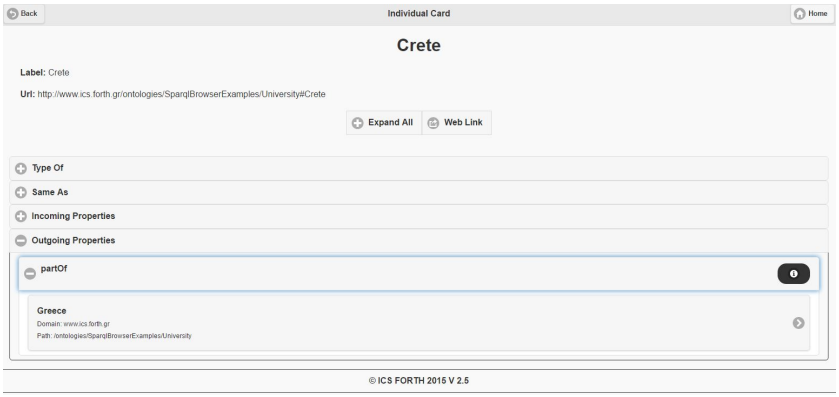

Figure 3.19: Individual card Screen - Crete - Outgoing properties

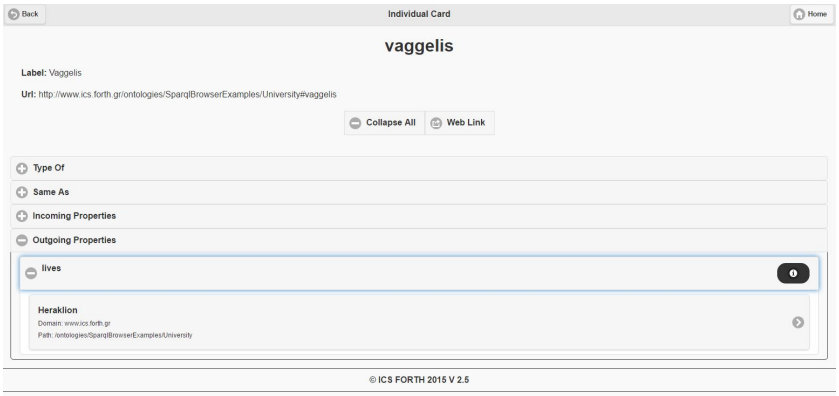

Figure 3.20: Individual card Screen - Vaggelis

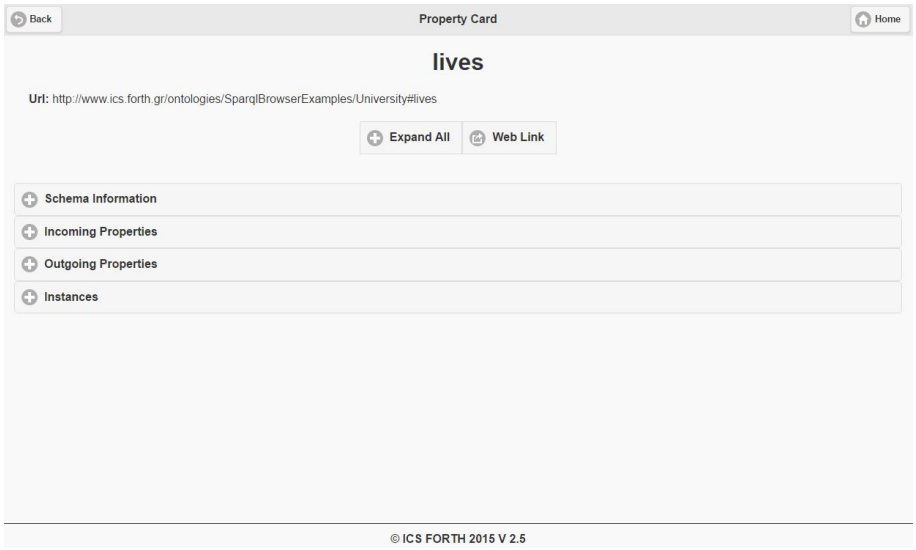

Figure 3.21: Property card Screen - lives

Chapter 4

Caching Approaches

Here we describe the client side caching approaches. At first we describe the the caching approaches (4.1), then (4.2) describe the cache refresh policies and finally (4.3) we provide the adopted caching mechanism

4.1 Approaches

The first rising question is what to cache? There are various options. Below we list four basic approaches:

(**a**) **AllLocal** The cache is a relational table storing the entire contents of the SPARQL endpoint. If the entire contents can fit to the cache size, then the table could store all triples and then the query language of the Web SQL database can be used for getting the required data.

(**b**) **FreqData** The cache contains the answers of a set of predefined and commonly used queries (e.g. the set of all classes, the set of all properties, etc). In this case the cache is a set of (ListName, listOfResources). The listOfResources could be stored as an HTML string that contains the string that should be shown (it contains the list of URIs and their labels).

(**c**) **URI-based** In this case the cache is a set of (URI, string) pairs. The URI corresponds to a browsable element of the application (i.e. something for which an HTML page should be produced whose production requires submitting several SPARQL queries to the remote SE), while the string is the HTML string which is actually the entire contents of the page that corresponds to that particular URI.

(**d**) **Ad-Hoc Structure** In this case the cache can have any structure that seems convenient for using it (i.e. querying it) for answering the queries that the application requires. In comparison to (a), here we cannot store the entire contents of the SE. Approaches (b) and (c) are special cases of this case. An example of such a structure: If the application very commonly requires answering queries of the (pseudocode) form q = "`select` d `where (<URI>,` a`,` b`)(`b`,`c`,`d`)`", then we could have a cache having the form of a table with two columns (u,d) whose contents will be the result of the evaluation of the following query "`select` u`,`d `where (<URI>,` a`,` b`)(`b`,`c`,`d`)`". If this table is locally stored then instead of sending to the SE the query q, the code can answer it by using the local table, i.e. by evaluating over the local table the query "`select` d `where (<URI>,`d`)`". We could also say a (d)-cache also covers the case where a *view* of the contents of the SPARQL endpoint is stored, and whenever we want to evaluate a query, we check if we can evaluate it over the contents of the view (query answering using views is discussed in [24]).

The advantages and shortcoming of each one are described in Table 4.1. Hereafter we shall use SE for SPARQL Endpoint.

Table 4.1: Advantages and disadvantages of caching approaches

Approach	Advantages	Shortcomings
(**a**) AllLocal	• Only at the first time when the user connects to a SE, he has to wait for some time. After this loading and local storage of the contents, the browsing is very fast since we don't have to send any SPARQL query to the SE. • The reliability of browsing is increasing. According to statistics, endpoints are often unavailable and have significant downtime, so this presents a serious obstacle in application development and scenarios which rely on the data.	• Feasible only if the entire contents of the SE fits to the cache size. • A database schema has to be defined for the local DB and all queries (for producing the pages) should not be written in SPARQL but in SQL which is presumed to be the default database schema.
(**b**) FreqData	• Some frequently used pages (listing of classes, instance, etc) are instantly available.	• The cache does not offer any speedup to the computation of the resource's cards. The computation of each resource card requires sending several queries to the SE.
(**c**) URI–based	• If the user has visited the card of a resource, then in the next visits its page will be instantly available.	• No speedup gain if the user never visits again the card of a resource.
(**d**) Ad–Hoc Structure	• Can be exploited for answering more than one queries.	• It has to be designed based on the particular query requirements of the navigation. • Since SPARQL is not supported in the local DB the rest js code should formulate queries over that structure.

4.2 Cache Refresh

The contents of the SPARQL endpoint may be altered over time so a related question is when a cached entry should be refreshed. Various methods can be adopted, also determined by the cache type.

Table 4.2: Refreshing policy

Approach	Refreshing policy
(**a**) AllLocal	Periodically, or after user request, the cache is refreshed in one shot.
(**b**) FreqData	Again periodically, or after user request, this cache is refreshed in one shot.
(**c**) URI–based	Some very commonly used resources could live for ever in the cache. The remaining resources could use a LRU replacement policy. It also follows the **(b)** policy in the sense that can contain an element that has been deleted in the SE, therefore a click on that would lead to an empty result.
(**d**) Ad–Hoc Structure	The refresh policy of the cache could be the same as **(a)**.

4.3 The adopted Caching Mechanism

Based on the previous analysis we have designed a method that exploits a combination of the previous approaches.

After the user enters the address of a SE, the system first is by default set on FreqData caching method and makes queries of K triples representing Class, Properties and Individuals (i.e. number of stored triples, read the VoID [25] descriptions of the 3 SE if available). Note that if the corresponding lists cannot fit to the size allocated for FreqData method, then it caches only those that can fit. Then if the user select a specific URI see a detailed card uses URI-based caching method as follows:

All parts of the information that card require, are requested from the cache. The cache lookups the requested URI and if found it returns the

corresponding information. If not then it issues the required SPARQL queries to the remote source. After receiving this information, and if the cache is not full, it stores the data to the cache and returns the requested information to the caller. If however the cache is full, then we have to decide which element(s) of the cache is to be removed, freeing the space required for hosting the new data. One widely used policy is the LRU (Least Recently Used), which requires adding to each cache entry a field time expressing the last time that this entry was requested. The availability of this information allows the cache to locate and remove the oldest entry. Except from the LRU policy we periodically refresh the contents of the cache or by after user request. This periodic refresh define that the cached SE contents refresh, if they were cached longer that 2 days (48 hours).

Conclusively should be clarified that, if all contents of the remote SE are cached, then the system use the a combination of AllLocal method with FreqData method for initialization and URI-based method for card generation as described above.

Algorithm 1 The (c)-caching approach algorithm for Card Generation

1: **Input:** The client side database DB
2: **Input:** The selected card URI
3: **Output:** Card generation HTML page
4: $CardExist \leftarrow DB.Cards(URI)$
5: **if** $CardExist$ **then**
6: $Resources \leftarrow DB.selectResources(URI)$
7: $HTML \leftarrow GenerateHTML(Resources)$
8: **else**
9: **while** $JSONResources \cup DB > MAXSIZEOF(DB)$ **do**
10: delete oldest SE contents
11: **end while**
12: $DB.insert(JSONResources)$
13: $HTML \leftarrow GenerateHTML(JSONResources)$
14: **end if**

Algorithm 2 The (b)-caching approach algorithm for System Initialization

1: **Input:** The client side database DB
2: **Input:** The selected warehouse URL
3: **Output:** System initialization HTML page
4: $WarehouseExist \leftarrow DB.Warehouses(URL)$
5: **if** $WarehouseExist$ **then**
6: $CachedDateTime \leftarrow DB.getTime(URL)$
7: **if** $NowDateTime() - CachedDateTime \geq 48hours$ **then**
8: $JSONResources \leftarrow HTTPGetInitializationResources()$
9: $DB.delete(URL)$
10: **while** $JSONResources \cup DB > MAXSIZEOF(DB)$ **do**
11: delete oldest SE contents
12: **end while**
13: $DB.insert(JSONResources)$
14: $HTML \leftarrow GenerateHTML(JSONResources)$
15: **else**
16: $Resources \leftarrow DB.selectResources(URL)$
17: $HTML \leftarrow GenerateHTML(Resources)$
18: **end if**
19: **else**
20: $JSONResources \leftarrow HTTPGetInitializationResources()$
21: **while** $JSONResources \cup DB > MAXSIZEOF(DB)$ **do**
22: delete oldest SE contents
23: **end while**
24: $DB.insert(JSONResources)$
25: $HTML \leftarrow GenerateHTML(JSONResources)$
26: **end if**

Chapter 5

Implementation and Application

At first (chapter 5.1) we provide details for the implementation and applicability details, then (chapter 5.2) we describe how the cache mechanism was implemented and finally (chapter 5.3) explore some difficulties that we encountered and how we eventually tackled them.

5.1 Used Libraries and Applicability

The PascoLink web project[1] is using various libraries and frameworks as follows:

The JQuery Mobile which is a touch-optimized HTML5 UI framework designed to make responsive web sites and apps that are accessible on every smart phone, tablet and desktop device.

The Web SQL Database which is a web page API for storing data locally at client side in databases that can be queried using a variant of SQL (the use SQL language dialect is SQLite 3.6.19).

The PhoneGap framework[2] and the nwjs project[3] which are open source solutions for building cross-platform hybrid mobile and desktop apps with standards-based Web technologies like HTML5, JavaScript, CSS.

The application can ran in the following platforms and browsers: For desktop browsers:

[1] http://www.ics.forth.gr/isl/PascoLink
[2] http://phonegap.com/
[3] https://github.com/nwjs

Table 5.1: Applicability on desktop browsers

Browsers	Chrome	Safari	Opera
Min Version	4	3.1	11.5
Max Version	48+	9+	33+

For mobile browsers:

Table 5.2: Applicability on mobile browsers

Browsers	Chrome for Android	iOS Safari	Android Browser
Min Version	-	3.2	2.1
Max Version	44+	9+	44+

5.2 Cache Implementation

Below we describe the pilot implementation of the (a) AllLocal method.

5.2.1 Pilot Phase

For caching the contents of the remote SE, we used the Web SQL Database based on SQLite that gives as all the power and effort of a structured SQL relational database. We created a database version 1.0 and granted database permission to scale up to the size of 5 MB. We periodically refresh the contents of the cache by deleting the SE contents that cached longer than 6 hours ago. If the cache size overcome the 5 MB, we defined a cache size recycle protocol. The protocol defines that after the overcome of the cache size we automatically delete the least showed cached SE contents. The database comprises twenty six tables which are used for caching the fetched RDF triples of remote SPARQL warehouses.

We have created eight tables in order to initialize the system. Firstly we store the endpoints URL, system configurations (language, Graph, etc.), the last time (hours) that user browsed the endpoint and the number of classes, properties and individuals of selected warehouse. After requesting the warehouse for getting a specific number of classes, properties and individuals we store information in the appropriate table representing every resource URI, label and description.

Then we created also eighteen tables that store resource's card details. The card details contains the schema information, incoming properties, outgoing properties and instances of every selected resource, incoming and outgoing property.

In particular:

1. The table **Endpoints** stores the id of each warehouse that has an unique auto generated value, the url address, the total amount of Classes, Properties and Individuals of the warehouse, the specified graph, the last time (hours) that user browsed the endpoint and the offset, limit of the requested triples of the remote endpoint.

2. The table **SearchedKeywords** stores the searched keyword id with an unique auto generated value, the id of the endpoint as a reference and the searched keywords as text.

3. The table **Subjects** stores the subjects id with an unique auto generated value, the id of the searched keyword as a reference and the URI.

4. The table **Predicates** stores the predicates id with an unique auto generated value, the id of the searched keyword as a reference and the URI.

5. The table **Objects** stores the objects id with an unique auto generated value, the id of the searched keyword as a reference and the URI.

6. The table **Classes** stores the classes id with an unique auto generated value, the id of the endpoint as a reference, the URI, the label, the description and information about when specific details card of the class is cached.

7. The table **Properties** stores the classes id with an unique auto generated value, the id of the endpoint as a reference, the URI, the label, the description and information about when specific details card of the property is cached.

8. The table **Individual** stores the classes id with an unique auto generated value, the id of the endpoint as a reference, the URI, the label, the description and information about when specific details card of the individual is cached.

9. The table **IncomingProperties** stores the incoming properties of entities using the id of each entity as a reference, the URI, the label and the text category (class,property,individual) of each entity as a reference.

10. The table **OutgoingProperties** stores the outgoing properties of entities using the id of each entity as a reference, the URI, the label and the text category (class,property,individual) of each entity as a reference.

11. The table **IncomingPropertiesInstances** stores the instances for each of the incoming properties of an entity and specifically the id of the Incoming property as a reference, the URI and the label.

12. The table **OutgoingPropertiesInstances** stores the instances for each of the outgoing properties of an entity and specifically the id of the Outgoing property as a reference, the URI and the label.

13. The table **Type** stores information about the type of each individual URI that composing the general schema information and specifically the id of the individual as a reference, the URI and the label.

14. The table **SameAs** stores information about the identical individual of each individual URI that composing the general schema information and specifically the id of the individual as a reference, the URI and the label.

15. The table **EquivalentTo** stores information about the equivalent classes of each class URI that composing the general schema information and specifically the id of the class as a reference, the URI and the label.

16. The table **DisjointWith** stores information about the disjoint classes of each class URI that composing the general schema information and specifically the id of the class as a reference, the URI and the label.

17. The table **Superclasses** stores information about the super classes of each class URI that composing the general schema information and specifically the id of the class as a reference, the URI and the label.

18. The table **Subclasses** stores information about the subclasses of each property URI that composing the general schema information and specifically the id of the class as a reference, the URI and the label.

19. The table **Domain** stores information about the domain of each property URI that composing the general schema information and specifically the id of the property as a reference, the URI and the label.

20. The table **Range** stores information about the range of each property URI that composing the general schema information and specifically the id of the property as a reference, the URI and the label.

21. The table **Subproperties** stores information about the sub properties of each property URI that composing the general schema information and specifically the id of the property as a reference, the URI and the label.

22. The table **Superproperties** stores information about the super properties of each property URI that composing the general schema information and specifically the id of the property as a reference, the URI and the label.

23. The table **DirectInstances** stores the direct instances of each class URI and specifically the id of class as a reference, the URI and the label.

24. The table **AllInstances** stores all instances of each class URI using inference and specifically the id of class as a reference, the URI and the label.

25. The table **ObjectInstances** stores the object instances of each property URI and specifically the id of property as a reference, the URI and the label.

26. The table **SubjectInstances** stores the subject instances of each property URI and specifically the id of property as a reference, the URI and the label.

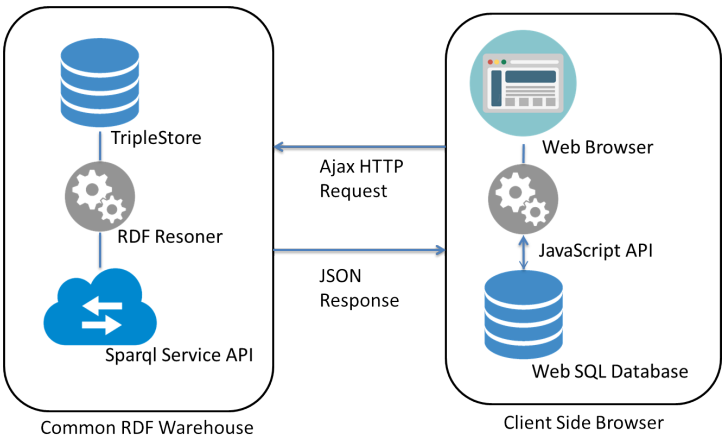

Figure 5.1: Client-side SPARQL browser implementation

As regards the flow of control, after the user selects or provides the URL of a specific remote SE, we send three HTTP Get requests to the remote endpoint and receive the URIs of Classes, Properties and Individuals in order to initialize the system. Then we used asynchronous repetition of requests to the warehouse to receive information regarding the label and the description of each URI. We used this technique in order to get label and description for

two reasons. The first is a limitation of SPARQL 1.0 version of getting only one label and comment of each resource. The second main reason is that when we used only one query to get each resource along with label and description we confronted time-out errors from remote endpoints with large size of contents. After the initialization process, user can select, search or request more results among the received classes, properties or individuals. When the user select a unique entry, an informational card will appear representing each URI. In this card user navigate through a more detailed information referred to URI.

5.2.2 Final Phase

(a)/(b): System Initialization (Eight tables)

- Endpoints(id, URI, Graph, lastViewTime, DisplayLanguage, countClasses, countProperties, countIndividuals, sparqlVersion)
- SearchedKeywords(id, endpointID, Keyword)
- Classes(id, endpointID, URI, label, Description)
- Properties(id, endpointID, URI, label, Description)
- Individuals(id, endpointID, URI, label, Description)
- Subjects(searchedkeywordsID, endpointID, URI)
- Predicates(searchedkeywordsID, endpointID, URI)
- Objects(searchedkeywordsID, endpointID, URI)

(a)/(c): Resource Card Generation (Eighteen tables)

- Subclasses(id, classID, endpointID, URI)
- EquivalentTo(id, classID, endpointID, URI)
- Superclasses(id, classID, endpointID, URI)
- DisjointWith(id, classID, endpointID, URI)
- Domain(id, propertyID, URI)
- Range(id, propertyID, URI)
- Subproperties(id, propertyID, URI)

- Superproperties(id, propertyID, URI)
- ObjectInstanes(id, propertyID, URI)
- SubjectInstanes(id, propertyID, URI)
- DirectInstances(id, classID, URI)
- AllInstances(id, classID, URI)
- Type(id, individualID, URI)
- SameAs(id, individualID, URI)
- IncomingProperties(id, endpointID, URI, Category)
- OutgoingProperties(id, endpointID, URI, Category)
- IncomingPropertiesInstances(id, endpointID, incomingPropertiesID,Category)
- OutGoingPropertiesInstances(id, endpointID, outgoingPropertiesID,Category)

5.3 Difficulties that we Encountered

In order to develop a client side browsing system we encountered some difficulties that we had to tackle. A difficulty that we encountered was the response times for keyword searching. Most SPARQL endpoints with large size of contents consume a lot of time in order to respond or send time-out errors. As a result we had to create controls and limits on our system and queries to save time. Then we had difficulties exporting response times for evaluation purposes. The local databases and Ajax requests don't have an API control for exporting the execution time. As a result we had to calculate results with timers on synchronous and asynchronous functions.

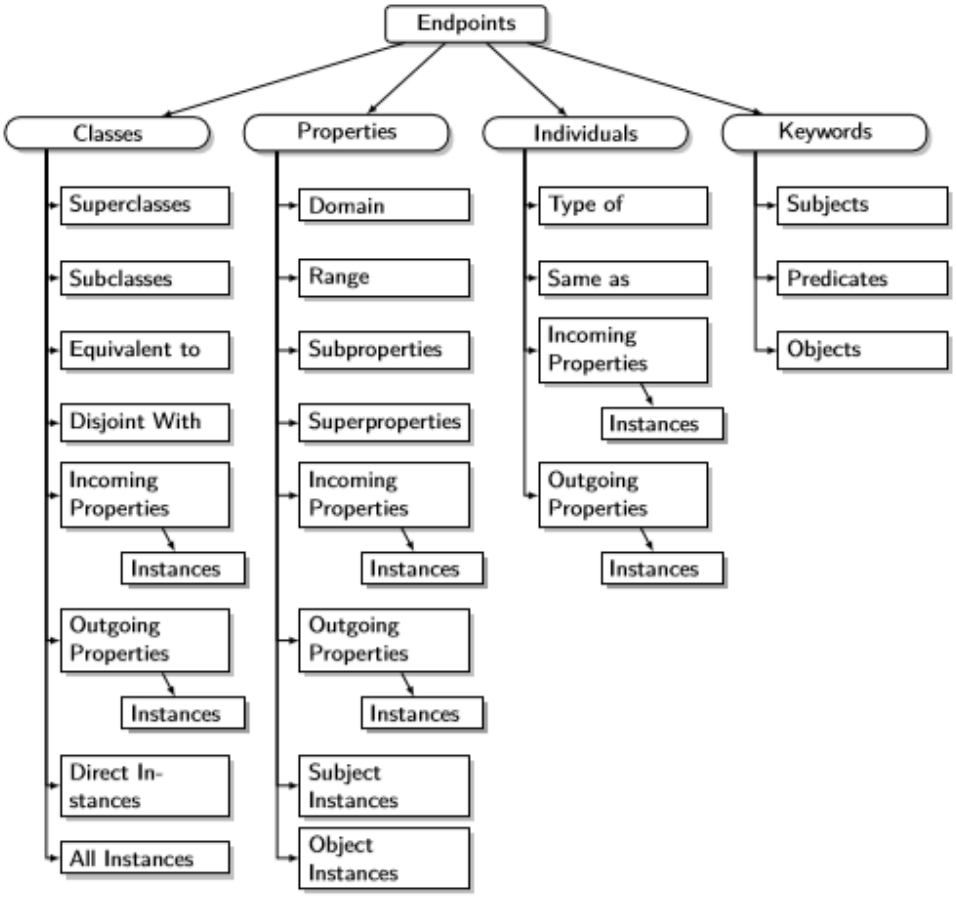

Figure 5.2: Database diagram

5.4 How to use

We implemented an integrated system for browsing SPARQL endpoints. We elaborated on how one can use his/her internet browser to scan through the contents of a remote SPARQL endpoint. To reach this objective we investigated a client-side approach that requires only a web browser and it's directly applicable over any SPARQL endpoint without any deployment or operational maintenance.

We implemented a web project named PascoLink. To use to this project a user has to visit the online PascoLink platform[4]. The PascoLink platform offers the opportunity for users to use the project with the use of a desktop or

[4]http://www.ics.forth.gr/isl/PascoLink

mobile browsers[5]. The user could also download the PascoLink project build for desktop and mobile devices. The desktop versions consist of Windows x64 and Mac x64 pre build version that user can download and follow the standard software installation process for each version. Furthermore, the mobile versions consist of Android (Fig. 5.3) and iPhone (Fig. 5.4) devices versions that are hosted on Play Store (build for bigger of equal API 19 platform versions) and App Store (build with iOS 8), respectively.

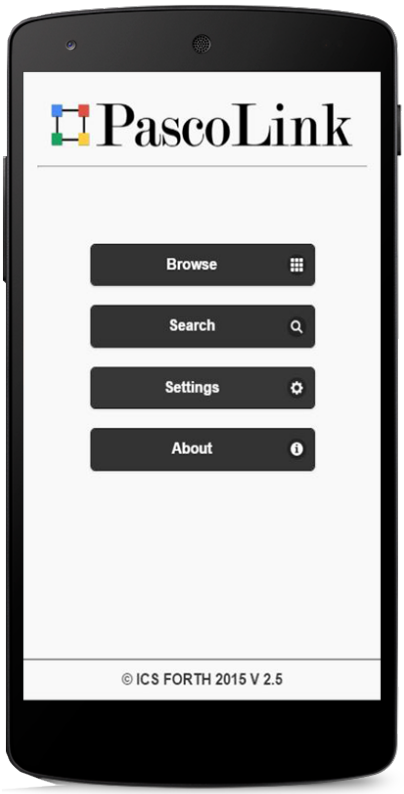

Figure 5.3: PascoLink on Nexus6x

[5]http://www.ics.forth.gr/isl/PascoLink/Browser

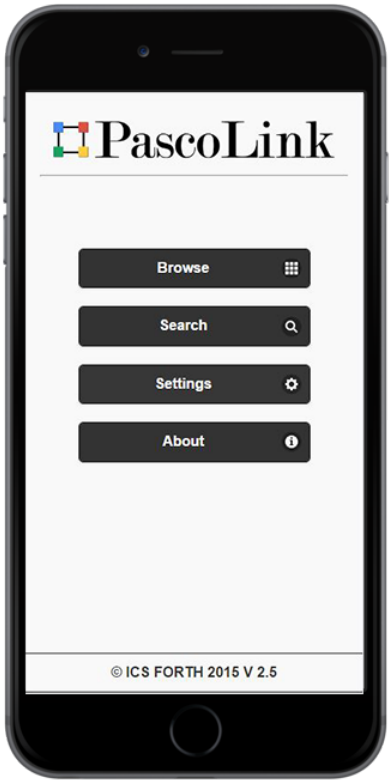

Figure 5.4: PascoLink on iPhone6

Chapter 6

Experimental Evaluation of the Cache Performance

In this chapter, firstly we describe the measures (section 6.1) and metrics (section 6.2) in order to execute the experimental evaluation. Then we provide a series of requests (section 6.4) and SPARQL endpoints (section 6.3) that were used and finally we report (section 6.5) and summarise (section 6.6) the results over the cache.

6.1 Measures

In order to make the results comparable and measure the efficiency of the caching mechanisms we used two measures:

- **Capacity**
  This measure represents the number (Query limit) of Classes, Properties and Individuals, resources that are being extracted with a SPARQL query through the remote warehouse. As a result, this metric is crucial because it affects the cache size.

- **Response time**
  This measure represents the time spent for initialization, selection, insertion or generation of resource card to complete in seconds.

All benchmarking was done on a machine with the following configuration: Intel Core 2 Quad (Q6600 4x2.40GHz), 2x2Gb of Ram, 250Gb SATA HD (7.2000rpm), Windows 7 64bit, Google Chrome 45.0.2454.85 m.

6.2 Metrics

Suppose that we want to comparatively evaluate two or more methods. We can compare them according to the following metrics:

- **Cache selection time**
  It is the time required from cache to retrieve (b)-cache information for different Capacity values.
- **Initialization time**
  It is the time required only at the beginning of the application that the system consumes in order to request resources and get a response from the remote endpoint, filling and retrieving the cached resources, until information is displayed to user for different Capacity values.
- **Cache insertion time**
  It is the time required from (b)-cache to store information for different Capacity values.
- **Average time required to compute one resource's card**
  This is the most important metric since the primary scenario is that of browsing. It is the average time that the system consumes in order to generate a detailed card about a resource. The detailed cards divided into Class, Property or Individual card.

6.3 Used SPARQL endpoints

We used the following SEs:

- The SE of ToyExample[1], which is a small and simple custom made semantic warehouse (52 triples and support of SPARQL 1.1 version) with information structured in such a way as to assist in better understanding the functionality of the Browser.
- The SE of Fishbase[2], which is a domain specific semantic warehouse (approximately 8.15 million triples and support of SPARQL 1.0 version) with information about global fish species.
- The SE of DBpedia[3], which is an online Triple store with structured content (approximately 438 million triples and support of SPARQL 1.1 version) extracted from Wikipedia.

[1] http://www.ics.forth.gr/isl/PascoLink/Endpoints/toyexample/index.html
[2] http://www.ics.forth.gr/isl/PascoLink/Endpoints/fishbase/index.html
[3] http://dbpedia.org/sparql

- The SE of the MarineTLO-based warehouse[4], which is a domain specific semantic warehouse (approximately 5.5 million triples and support of SPARQL 1.0 version) with information about the marine domain.

6.4 Series of Requests

To compute average times we should use a series of requests. They can be custom, random, synthetically produced, or stem from real log files (e.g. the query logs used in [23]). In our case we used custom query requests to initialize system or generate resource card information. For initialization purposes we separated the use of the queries according to the SPARQL version (1.0 or 1.1) that the remote endpoint support in order to use the abilities of each SPARQL API documentation. As a result, we send a request with a simple SPARQL 1.1 query (Chapter 6.4.1.1) before initialization to the remote endpoint. When the response is successful means that the remote endpoint supports 1.1 version, otherwise it supports 1.0 version. Then we used 4 queries, requesting Classes (Chapter 6.4.1.2 or Chapter 6.4.1.3), Properties (Chapter 6.4.1.4 or Chapter 6.4.1.5), Individuals (Chapter 6.4.1.6 or Chapter 6.4.1.7) and each resource label and description (Chapter 6.4.1.8). We sent 3 requests for Classes, Properties and Individuals and for each resource we send a request for one label and description. As a result we used asynchronous repetition of requests that equals the total number of Classes, Properties and Individuals resources to the warehouse to receive information regarding the label and the description of each URI. We used this technique in order to get label and description for two reasons. The first is a limitation of SPARQL 1.0 version of getting only one label and comment of each resource. The second main reason is that when we used only one query to get each resource along with label and description we confronted time-out errors from remote endpoints with large size of contents.

The generation of a detailed card is divided into Class (Chapter 6.4.2.1), Property (Chapter 6.4.2.2) or Individual card (Chapter 6.4.2.3). For each card we retrieve information regarding: Schema information, instances (The AllInstances and DirectInstances for Class card and Object, Subject instances for Property card), incoming (Chapter 6.4.2.4), outgoing properties (Chapter 6.4.2.5) and the instances of incoming (Chapter 6.4.2.6) and outgoing properties (Chapter 6.4.2.7). We used 7 queries for Class card, 7 for Property card and 5 for Individual card generation. We sent 5 requests for Class card (Schema information, Incoming properties, Outgoing properties,

[4]https://virtuoso.i-marine.d4science.org:4443/sparql

All Instances, Direct Instances), 5 for Property card (Schema information, Incoming properties, Outgoing properties, Object Instances, Subject Instances) and 3 for Individual card (Schema information, Incoming properties, Outgoing properties). For generating any card, we request the instances of incoming and outgoing properties. The number of these requests is dynamic and depends on the number of incoming and outgoing properties of the selected resource.

We created an online platform[5] that presents all these queries in every query category. We also describe the above queries in the following sub sections.

6.4.1 System Initialization Queries

6.4.1.1 SPARQL version of remote endpoint

```
ask {?a a <dymmy> FILTER(STRSTARTS(STR(?a), 'dummy'))}
```

6.4.1.2 Classes - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?class
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
        ?class!=owl:FunctionalProperty &&
        ?class!=owl:disjointWith&&
        ?class!=owl:AnnotationProperty &&
        ?class!=owl:InverseFunctionalProperty &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:SymmetricProperty &&
```

[5] http://www.ics.forth.gr/isl/PascoLink/bench/Queries/queries.html

```
        ?class!=owl:DeprecatedClass &&
        ?class!=owl:DeprecatedProperty &&
        ?class!=owl:DataRange &&
        ?class!=owl:DatatypeProperty &&
        ?class!=owl:Ontology &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:Thing &&
        ?class!=owl:Restriction &&
        ?class!=owl:ObjectProperty &&
        ?class!=owl:Nothing &&
        ?class!=owl:AllDifferent &&
        ?class!=owl:NamedIndividual &&
        ?class!=owl:Class &&
        ?class!=owl:OntologyProperty &&
        ?class!=rdfs:Class &&
        ?class!=rdf:Property &&
        ?class!=rdf:List &&
        ?class!=rdfs:ContainerMembershipProperty &&
        ?class!=rdfs:Container &&
        ?class!=rdfs:Literal &&
        ?class!=rdfs:Datatype &&
        ?class!=rdfs:Resource &&
        ?class!=rdf:Statement &&
        ?class!=rdf:Alt &&
        ?class!=rdf:Seq &&
        ?class!=rdf:Bag &&
        ?class!=rdf:XMLLiteral &&
        !isBlank(?class)
)} OFFSET 0 LIMIT 100
```

6.4.1.3 Classes - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?class
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
```

```
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
   !STRSTARTS(STR(?class),'http://www.w3.org/2002/07/owl')&&
   !STRSTARTS(STR(?class),'http://www.w3.org/2000/01/rdf-schema')&&
   !STRSTARTS(STR(?class),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
   !isBlank(?class)
)
} OFFSET 0 LIMIT 100
```

6.4.1.4 Properties - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
        ?prop!=owl:sameAs &&
        ?prop!=owl:differentFrom &&
        ?prop!=owl:versionInfo &&
        ?prop!=owl:priorVersion &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:oneOf &&
        ?prop!=owl:unionOf &&
        ?prop!=owl:complementOf &&
```

```
        ?prop!=owl:hasValue &&
        ?prop!=owl:disjointWith &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:allValuesFrom &&
        ?prop!=owl:cardinality &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:distinctMembers &&
        ?prop!=owl:equivalentClass &&
        ?prop!=owl:equivalentProperty &&
        ?prop!=owl:imports &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:intersectionOf &&
        ?prop!=owl:inverseOf &&
        ?prop!=owl:maxCardinality &&
        ?prop!=owl:minCardinality &&
        ?prop!=owl:onProperty &&
        ?prop!=owl:someValuesFrom &&
        ?prop!=rdfs:member &&
        ?prop!=rdfs:range &&
        ?prop!=rdfs:domain &&
        ?prop!=rdf:type &&
        ?prop!=rdfs:subClassOf &&
        ?prop!=rdfs:subPropertyOf &&
        ?prop!=rdfs:label &&
        ?prop!=rdfs:comment &&
        ?prop!=rdf:subject &&
        ?prop!=rdf:predicate &&
        ?prop!=rdf:object &&
        ?prop!=rdfs:seeAlso &&
        ?prop!=rdfs:isDefinedBy &&
        ?prop!=rdf:first &&
        ?prop!=rdf:rest &&
        ?prop!=rdf:nil &&
        ?prop!=rdf:value
)} OFFSET 0 LIMIT 100
```

6.4.1.5 Properties - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
SELECT DISTINCT ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
   !STRSTARTS(STR(?prop),'http://www.w3.org/2002/07/owl')&&
   !STRSTARTS(STR(?prop),'http://www.w3.org/2000/01/rdf-schema')&&
   !STRSTARTS(STR(?prop),'http://www.w3.org/1999/02/22-rdf-syntax-ns')
)
} OFFSET 0 LIMIT 100
```

6.4.1.6 Individuals - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?uri
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
}
UNION{
        ?uri rdf:type owl:NamedIndividual
}
FILTER (
        ?y!=owl:FunctionalProperty &&
        ?y!=owl:AnnotationProperty &&
        ?y!=owl:InverseFunctionalProperty &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:SymmetricProperty &&
```

```
        ?y!=owl:DeprecatedClass &&
        ?y!=owl:DeprecatedProperty &&
        ?y!=owl:DataRange &&
        ?y!=owl:DatatypeProperty &&
        ?y!=owl:Ontology &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:Thing &&
        ?y!=owl:Restriction &&
        ?y!=owl:ObjectProperty &&
        ?y!=owl:Nothing &&
        ?y!=owl:AllDifferent &&
        ?y!=owl:NamedIndividual &&
        ?y!=owl:Class &&
        ?y!=owl:OntologyProperty&&
        ?y!=rdfs:Class &&
        ?y!=rdf:Property &&
        ?y!=rdf:List &&
        ?y!=rdf:Alt &&
        ?y!=rdf:Seq &&
        ?y!=rdf:Bag &&
        ?y!=rdfs:ContainerMembershipProperty &&
        ?y!=rdfs:Container &&
        ?y!=rdfs:Literal &&
        ?y!=rdfs:Datatype &&
        ?y!=rdfs:Resource &&
        ?y!=rdf:Statement &&
        ?y!=rdf:XMLLiteral &&
        !isBlank(?uri)
)} OFFSET 0 LIMIT 100
```

6.4.1.7 Individuals - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?uri
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
```

```
}
UNION{
        ?uri rdf:type owl:NamedIndividual
}
FILTER (
   !STRSTARTS(STR(?y),'http://www.w3.org/2002/07/owl')&&
   !STRSTARTS(STR(?y),'http://www.w3.org/2000/01/rdf-schema')&&
   !STRSTARTS(STR(?y),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
   !isBlank(?uri)
)
} OFFSET 0 LIMIT 100
```

6.4.1.8 Label and Description

```
SELECT ?label ?comment
FROM <Graph name if selected>
WHERE {
   {
        <Resource URI> rdfs:label ?label
   }
   UNION{
        <Resource URI> rdfs:comment ?comment
   }
} LIMIT 2
```

6.4.2 Card Generation Queries

6.4.2.1 Class

Schema information

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?subclass ?equiv ?disjoint ?superclass
FROM <Graph name if selected>
WHERE {
  {
        ?subclass rdfs:subClassOf <Resource Class URI>
  }UNION{
        <Resource Class URI> owl:equivalentClass ?equiv
  }UNION{
```

```
        <Resource Class URI> owl:disjointWith ?disjoint
 }UNION{
        <Resource Class URI> rdfs:subClassOf ?superclass
 }
}
```

Direct instances

```
SELECT DISTINCT ?y
FROM <Graph name if selected>
WHERE {
        ?z ?y <Resource Card URI>
}
```

All instances

```
define input:inference "Inference_Rule_if_selected"
SELECT DISTINCT ?x
FROM <Graph name if selected>
WHERE {
        ?x a <Resource Card URI>
}OFFSET 0 LIMIT 100
```

6.4.2.2 Property

Schema information

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?domain ?range ?subproperty ?superproperty
FROM <Graph name if selected>
WHERE {
 {
        <Resource Property URI> rdfs:subPropertyOf ?subproperty
 }UNION{
        <Resource Property URI> rdfs:domain ?domain
 }UNION{
        <Resource Property URI> rdfs:range ?range
 }UNION{
        ?superproperty rdfs:subPropertyOf <Resource Property URI>
 }
}
```

Subject instances

```
SELECT DISTINCT ?a
FROM <Graph name if selected>
WHERE {
        ?a <Resource Property URI> []
}OFFSET 0 LIMIT 100
```

Object instances

```
SELECT DISTINCT ?a
FROM <Graph name if selected>
WHERE {
        [] <Resource Property URI> ?a
}OFFSET 0 LIMIT 100
```

6.4.2.3 Individual

Schema information

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?typos ?sameas
FROM <Graph name if selected>
WHERE {
 {
        <Resource Individual URI> rdf:type ?typos
 }UNION{
        <Resource Individual URI> owl:sameAs ?sameas
 }
}
```

6.4.2.4 Incoming properties

```
SELECT DISTINCT ?y
FROM <Graph name if selected>
WHERE {
        ?z ?y <Resource Card URI>
}
```

6.4.2.5 Outgoing properties

```
SELECT DISTINCT ?y
FROM <Graph name if selected>
WHERE {
        <Resource Card URI URI> ?y ?z
}
```

6.4.2.6 Instances of Incoming properties

```
SELECT DISTINCT ?instances
FROM <Graph name if selected>
WHERE {
        ?instances <Incoming Property URI> <Resource Card URI>
}
```

6.4.2.7 Instances of Outgoing properties

```
SELECT DISTINCT ?instances
FROM <Graph name if selected>
WHERE {
        <Resource Card URI> <Outgoing Property URI> ?instances
}
```

6.5 Carried out Experiments

To automatically evaluate the cache performance with real time data of three online warehouses as described above, we create an online evaluation system[6].

The experiments relate to the system initialization time (Fig. 6.1), the selection time (Fig. 6.2), the insertion time (Fig. 6.3) and the resource card generation time (Fig. 6.4). To produce the final results we run all experiments in every remote endpoint for five distinct Capacity values (50, 100, 200, 500 and 1000) and then calculate the average time for every experimental procedure. We represent Capacity values as a number (Query limit) of Classes, Properties and Individuals; resources that are being extracted with a SPARQL query through the remote endpoint that affect the cache size.

We used this metric (Capacity) to represent the cache size for two reasons. The first is a limitation of the Web SQL Database API in getting the database (cache) size in MB. We tried to overcome this limitation and estimate the cache size in MB for every Capacity value from the web browser's database file, in order to understand the way that the Capacity affects the cache size in MB. As a result, we estimate that when the Capacity is 50, 100, 200, 500 or 1000 the cache size is approximately 0.14 MB, 0.17 MB, 0.22 MB, 0.34 MB and 0.78 MB, respectively. The second main reason is that the memory of the used local storage technique is shared across every window or tab on the running web browser. As a result, the size of the memory is affected by every application or web page that uses client side storage.

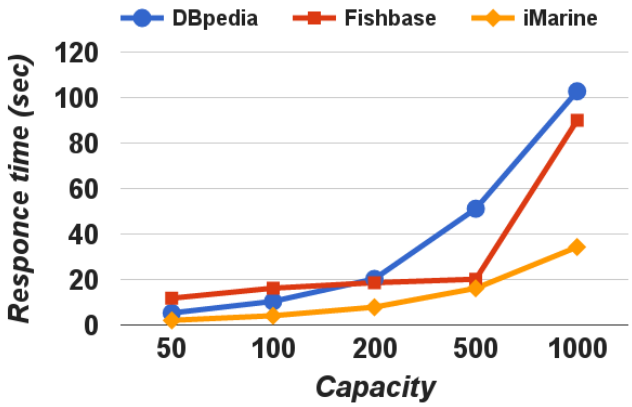

Figure 6.1: (b)-caching approach - Initialization time

[6]http://www.ics.forth.gr/isl/PascoLink/bench/index.html

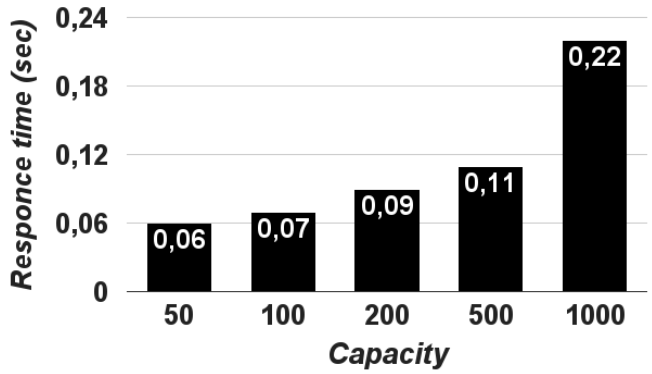

Figure 6.2: (b)-caching approach - Cache selection time.

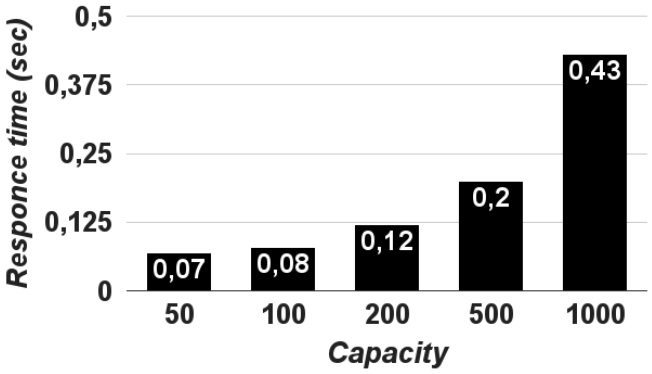

Figure 6.3: (b)-caching approach - Cache insertion time.

The experimental result in the initialization (Fig. 6.1) of the (b)-caching approach represents the time in seconds that the system consumes in order to be initialized with different Capacity values with the use of (b)-cache. In order to export these results we measure time between requesting (Chapter 6.4) Classes (Chapter 6.4.1.2), Properties (Chapter 6.4.1.4), Individuals (Chapter 6.4.1.6) and each resource label and description (Chapter 6.4.1.8), through remote endpoint, filling and retrieving these resources from the cache, until information is displayed to the user. The initialization response time increases when the Capacity values or the size of the contents of the remote SE are being increased. The reason is that Capacity defines the number of the resources that are being retrieved from a remote endpoint. As a result, it

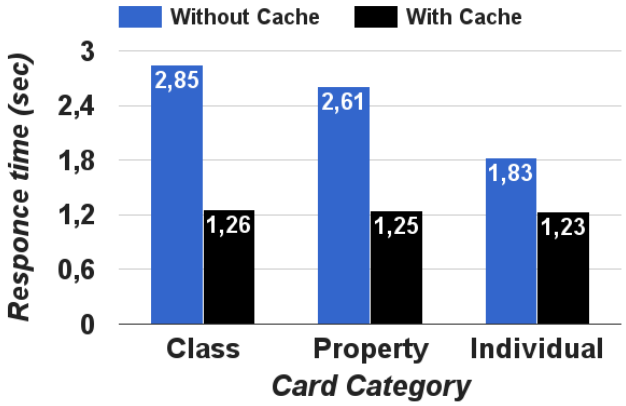

Figure 6.4: (c)-caching approach - Card creation time.

not only affects the response time that the remote endpoint needs, in order to send the requested resources, but also the response time of the (b)-cache to insert and retrieve the initialization resources. Furthermore, the size of the contents on the remote endpoint also affect the correspondence time that the remote endpoint needs, in order to return the requested resources by increasing the system's initialization response time.

The selection time result (Fig. 6.2) represents time in seconds that the system consumes in order to select the cached initialization resources from (b)-cache to the user with different cache sizes, defined by the Capacity values. The selection response time elevates when the Capacity values are being increased. This is because Capacity affects the number of the cached resources that are being retrieved from (b)-cache.

After the initialization and selection time finish we measure the time that is required to store data to (b)-cache for different amounts of data defined by Capacity values (Fig. 6.3). The cache insertion time ranges from 0.07 seconds for 50 Capacity to 0.22 seconds for a 1000 Capacity. The insertion time result has been taken into account in computing the initialization time that the system needs to display resources. The insertion response time increases when the Capacity values inflate. This is because the Capacity affects the number of resources that will be inserted to (b)-cache.

The times for the initialization ranges from 2.04 (50 Capacity) to 34.27 seconds (1000 Capacity) for the iMarine warehouse, from 5.27 (50 Capacity) to 102.76 seconds (1000 Capacity) for the DBpedia warehouse and from 11.77 (50 Capacity) to 89.95 (1000 Capacity) seconds for the Fishbase. To compute the average initialization time for every distinct Capacity value we sum the initialization times of DBpedia, Fishbase and iMarine with the same Capacity

Table 6.1: Initialization Speedup for (b)-cache

	Initialization time (sec)						
Capacity (Query limit)	DBpedia	Fishbase	iMarine	Average Initialization time (sec)	Average Selection time (sec)	Cache Speedup	Average Cache Speedup
50	5.27	11.77	2.04	6.36	0.06	99.05 %	99%
100	10.39	16.19	4.07	10.21	0.07	99.31 %	
200	20.25	18.59	7.84	15.56	0.09	99.42 %	
500	51.12	20.13	16.14	29.13	0.11	99.62 %	
1000	102.76	89.95	34.27	75.66	0.22	99.70 %	

and divide it by 3 (the number of remote endpoints). As a result, the average initialization response time for every Capacity are approximately 6, 10, 15, 29 and 75 seconds (Table 6.1).

If we compare the average initialization times (Table 6.1) with cache selection times (Fig. 6.2, Table. 6.1), we observe that when the initialization resources are cached, the system's initialization time (selection time) is much lower than the time between the request and response from a remote endpoint (average initialization time) for the different Capacity values. To compute the percentage of this speedup that (b)-cache offers on system's initialization time for every Capacity value we used this formula:

$$\frac{\text{Average initialization time} - \text{Average Selection time}}{\text{Average initialization time}}$$

As a result we computed that the (b)-cache speedup the initialization time by 99.05%, 99.31%, 99.42%, 99.62% and 99.70% for Capacity value 50, 100, 200, 500 and 1000, respectively. Taking into account the aforementioned speedup that (b)-cache offers to initialization time for the Capacity values, we can compute that the average speedup in the system's initialization time is 99% on average. We observe (Table 6.1) that the best values for Capacity are 50 and 100 in order to have the maximum human readable resources in a relatively short time (less than 10 seconds) along with the maximum cache speedup. We picked and configured our system with the best Capacity value (coincidentally also the maximum) that is 100 for requesting the remote endpoints.

The card generation result (Fig. 6.4) represents the time in seconds that the system consumed in order to create a detailed card (Chapter 6.4, 6.4.2.1, 6.4.2.2, 6.4.2.3, 6.4.2.4, 6.4.2.5, 6.4.2.6, 6.4.2.7) of a URI resource with and without the use of (c)-cache. In order to export this result we picked 1000 random URIs for every resource category (Class, Property, Individuals). Then we estimated the average response time that the system consumes in gen-

Table 6.2: Cache Card Generation Speedup for (c)-cache

Card Generation	Without Cache (sec)	With Cache (sec)	Cache Speedup	Average Cache Speedup
Class	2.85	1.26	55.78 %	46%
Property	2.61	1.25	52.10 %	
Individual	1.83	1.23	32.78 %	

erating the detailed card until the resource's card data is displayed to the user. The response time for generating the resource card for each of the classes, properties and individuals without the use of cache are 2.85, 2.61, 1.83 seconds and with the use of cache are 1.26, 1.25 and 1.23 seconds (Fig. 6.4).

If we compare, card generation times with the use of cache and without cache, we observe (Table 6.2) that when the information of the detailed card is cached (With Cache) the response time is lower than the generation time for a system request in constructing the detailed card and getting a response from a remote endpoint (Without Cache). To compute the percentage of this speedup that (c)-cache offers on card's generation time, we used this formula:

$$\frac{\text{Without Cache} - \text{With Cache}}{\text{Without Cache}}$$

As a result, we calculated that the (c)-cache speeds up the card generation time by 55%, 52% and 32% for the Class, Property and Individual cards respectively. Taking into account the above speedup that (c)-cache offers to the Card generation time in generating a detailed card, we can infer that the average speedup to the system's card generation time is 46% on average.

Table 6.3: Overall Cache Speedup

Caching Approach	Cache speedup	Overall Cache speedup
(b)	99%	73%
(c)	46%	

Generally, taking into account the extracted speedup that the (b) and (c) caching approach (Table. 6.3) offers on the system initialization and card generation time, we can conclude that the cache speeds up the browsing experience by 73% on average. This is theoretical approximation and it is not applicable to every possible scenario that may occur on a real time execution.

6.6 Synopsis of the Experimental Results

To resume, we executed an experimental evaluation over the adopted caching approach that we used to create the client side SPARQL browser. This experimental evaluation has revealed some useful caching efficiency results. In order to make these results comparable and measure the efficiency over the caching mechanisms we used two measures. The first is the Capacity that was used to represent the number (Query limit) of Classes, Properties and Individuals, resources that are being extracted with a SPARQL query through the remote warehouse. The second measure is the Response time that represents the time spent for initialization, selection, insertion or generation of a resource card to complete in seconds.

Then we used four metrics in order to comparatively evaluate two or more caching methods. The first is the cache selection time that (b)-cache needs in order to retrieve information. The second is the initialization time that represents the time required only at the beginning of the application that the system needs in order to request resources and get a response from the remote endpoint, filling and retrieving the cached resources for different Capacity values. The third metric is the cache insertion time that represents the time required from (b)-cache to store information for different Capacity values. The last metric is the card generation time that represents the average time that the system consumes in order to generate a detailed card about a resource, with and without the use of (c)-caching approach.

The experimental evaluation was executed upon four remote SPARQL endpoints. These remote endpoints were the ToyExample (52 triples), the Fishbase (approximately 8.15 million triples), the DBpedia (approximately 438 million triples) and the MarineTLO-based warehouse (approximately 5.5 million triples).

To compute the efficiency that the system achieves with the use of the cache, we had to use a series of requests that were used in order to browse the contents of the aforementioned SPARQL endpoints, by measuring the average execution times. These series of requests are custom query requests that are used to initialize the system or generate a resource card. For initialization purposes, we sent 3 requests for Classes, Properties and Individuals and for each resource we send a request for one label and description. Furthermore, the generation of a detailed card divided into Class, Property or Individual. We sent 5 requests for Class card (Schema information, Incoming properties, Outgoing properties, All Instances, Direct Instances), 5 for Property card (Schema information, Incoming properties, Outgoing properties, Object Instances, Subject Instances) and 3 for Individual card (Schema information, Incoming properties, Outgoing properties). For generating any card, we re-

quest the instances of incoming and outgoing properties. The number of these requests is dynamic and depends on the number of incoming and outgoing properties of the selected resource.

With the use of the experimental analysis related to the system initialization time (Fig. 6.1), selection time (Fig. 6.2), insertion time (Fig. 6.3) and resource card generation time (Fig. 6.4), the experiments have shown that the (b)-caching approach speeds up the system's initialization time approximately by 99% and the (c)-caching approach speeds up the generation of a detailed resource card by 46% in average. Generally, taking into account the aforementioned speedup that (b) and (c) caching approaches offer on the system's initialization time and the card generation time, we can conclude that the cache speeds up the browsing experience approximately by 73% on average. The cache speeds up the browsing experience regardless of the size of the contents of the remote endpoint, offering a smooth and fast browsing of any SPARQL endpoint without the creation of any server side implementation.

Chapter 7

Discussion

In section 7.1 we discuss how the cache could be exploited also by keyword search.

7.1 Querying

Let's discuss the more basic, and commonly used, requirement. The user would like to see whether the SPARQL endpoint contains information about a particular real world entity (entity in the broad sense), and therefore submits a keyword search query containing one or more words. He would get back the related (if any) resources, ideally a ranked list of resources starting from the more relevant ones. Then the user could select the desired and continue its browsing. This functionality prerequisites that at least substring matching should be supported.

Suppose the user types and submits a string 'Keyword'. The client then sends the following queries that return URIs and the total number of keyword appearances representing Subjects, Predicates, Objects as follows:

In particular:

- **Subjects**

```
SELECT DISTINCT ?s
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?s, "Keyword", "i") &&
                isIRI(?s)
```

```
        )
} OFFSET 0 LIMIT 20
```

- **Predicates**

```
SELECT DISTINCT ?p
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?p, "Keyword", "i") &&
                isIRI(?s)
        )
} OFFSET 0 LIMIT 20
```

- **Objects**

```
SELECT DISTINCT ?o
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?o, "Keyword", "i") &&
                isIRI(?s)
        )
} OFFSET 0 LIMIT 20
```

- **Total Subjects**

```
SELECT count(DISTINCT ?s)
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?s, "Keyword", "i") &&
                isIRI(?s)
        )
}
```

- **Total Predicates**

```
SELECT count(DISTINCT ?p)
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?p, "Keyword", "i") &&
                isIRI(?s)
        )
}
```

- **Total Objects**

```
SELECT count(DISTINCT ?o)
FROM <Graph name if selected>
WHERE {
        ?s ?p ?o
        FILTER (
                regex(?o, "Keyword", "i") &&
                isIRI(?s)
        )
}
```

The keyword searching requirement is also the subject of related server-based systems. In [26] the authors present an entity search that adapts a state-of-the-art IR ranking model by taking into consideration the structure and semantics of RDF data. [27] presents a novel form of language models for the structured, but schema-less setting of RDF triples and extended SPARQL queries with a ranking method that is based on statistical language models, a modern paradigm in information retrieval. [28] also introduces a novel keyword search paradigm for graph-structured data, focusing in particular on the RDF data model.

Even if the server-side provides techniques for implementing an effective and efficient keyword searching method as described above, the keyword search in some cases can be extremely slow. To measure the efficiency of keyword searching process of a remote endpoint, we used and searched a keyword named "albacares" and then we calculated the average time that every remote endpoint (Chapter 6.2) require to response. The "albacares"

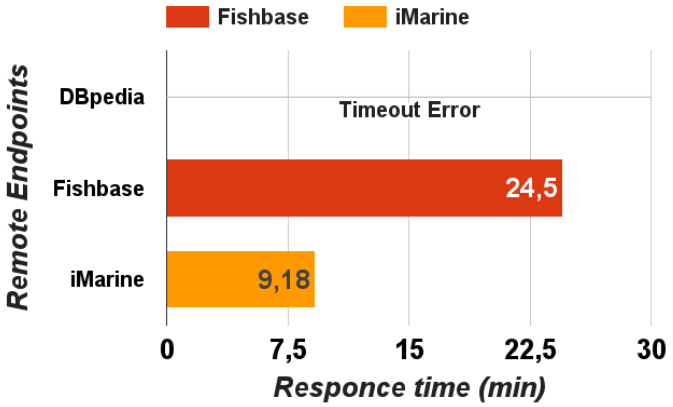

Figure 7.1: Keyword searching on subjects, predicates and objects

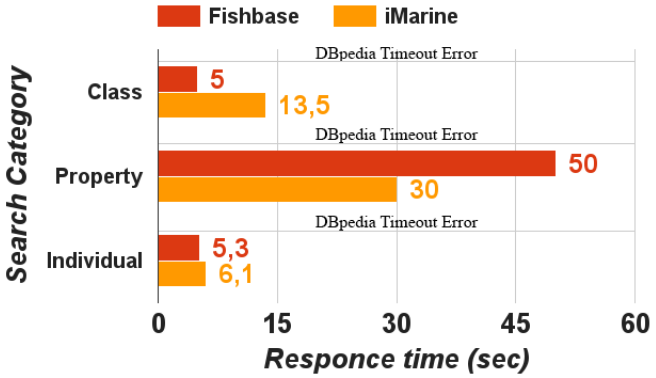

Figure 7.2: Keyword searching on classes, properties and individuals

keyword refers to the yellowfin tuna, that is a species of tuna found in pelagic waters of tropical and subtropical oceans worldwide.

Firstly, we searched the "albacares" keyword appearances representing Subjects, Predicates and Objects. We used 3 queries as described above, requesting Subjects, Predicates and Objects and then we measured the time that the remote endpoints require to response in minutes (Fig. 7.1). The response time for searching the keyword ranges from 24.5 minutes for Fishbase warehouse to 9.18 minutes for iMarine endpoint. The DBpedia warehouse encountered a timeout error for retrieving Subject, Predicates and Objects (Fig. 7.1). As a result, we observe that the keyword searching for Subject, Predicates and Objects is an extremely time consuming operation. The remote endpoints process a huge volume of data that contain RDF triples in

order to response and frequently lead to timeout errors.

Accordingly, the "albacares" keyword was searched for appearances representing Classes, Properties or Individuals. We used 3 queries (Appendix A.1) requesting Classes, Properties and Individuals. Each query divided in two queries and used accordingly to the SPARQL version (1.0 or 1.1) that the remote endpoint supports in order to use the abilities of each SPARQL API documentation. Then we measured the time that the remote endpoints require to respond in seconds (Fig. 7.2) for each entity category. The response time for searching the keyword on Classes, Properties and Individuals ranges from 5, 50 and 5.3 seconds for Fishbase warehouse to 13.5, 30 and 6.1 seconds for iMarine endpoint, respectively. The DBpedia warehouse encountered a timeout error for retrieving Classes, Properties and Individuals (Fig. 7.2). As a result, we observe that the keyword searching for Classes and Individuals response time is relatively short (less than 15 seconds). Additionally, the response time for keyword searching for Properties is a time consuming operation. The reason is that the remote endpoint has to parse all stored predicates that match the keyword. This parsing especially for huge warehouse creates significant delays to a remote endpoint response.

The keyword search results have shown that searching on a remote endpoint could be an operation that in some cases can be extremely slow. As a result, our system with the use of client side caching can also complement the server-side searching, and increase the overall efficiency as experienced by the user. Our system supports searching of Subjects, Predicates and Objects as described above and also keyword searching on Classes, Properties and Individuals. Then we describe how the system can speed up the keyword search. At first we would request the resources containing the keyword (if resources wasn't already cached), the remote endpoint would send a response and then we would store or retrieve the cached resources using the (b)-caching approach that was previously described, implemented and evaluated (Chapter 4.1, 5.2.2, 6.5).

Chapter 8

Concluding Remarks and Future work

There is already a plethora of SPARQL endpoints and their number keeps increasing. The amount of data to be managed is stretching the scalability limitations of SPARQL endpoints that are conventionally used to manage Semantic Web data. At the same time, the Semantic Web is increasingly reaching end users who need efficient and effective browsing of the contents of these queryable datasets and this is the reason why browsable HTML pages are also provided in many cases.

In this MSc thesis we analysed and implemented an integrated system for browsing SPARQL endpoints. We elaborated on how one can use his/her internet browser to scan through the contents of a remote SPARQL endpoint. To reach this objective we investigated a client-side approach that requires only a web browser and it's directly applicable over any SPARQL endpoint without any deployment or operational maintenance.

To maximize the utilization of the client's resources thus increasing the efficiency of browsing, we present how we can exploit the new features that HTML5 offers, providing a caching mechanism. We discuss the various approaches that could be used for caching and then we present and evaluate a sophisticated caching mechanism for the problem at hand.

To the best of our knowledge, this is the first work that focuses on a pure client-side solution for providing browsing of SPARQL endpoints that pays special attention to client caching by discussing various caching approaches, proposing a caching mechanism and finally experimentally evaluating the proposed caching mechanism. The distinctive characteristic of this project is that it offers a sophisticated caching mechanism. This is important not only for speeding up the browsing but also for alleviating the load of the SE. The application and the experimental results have shown that the (b)-

caching approach speeds up the system's initialization time approximately by 99% and that the (c)-caching approach speeds up the generation of a detailed resource card by 46% on average. Generally, taking into account the aforementioned speedup that (b) and (c) caching approach offers on system's initialization time (overhead) and card generation time. We can conclude that the cache speeds up the browsing experience approximately by 73% on average, offering a smooth and fast browsing of any SPARQL endpoint without the creation of any server side implementation.

There are many directions that we are currently exploring or plan to work in the immediate future. First of all, an important issue deserving further consideration is client side keyword search. Users in order to browse the contents of a remote endpoint overwhelmingly prefer imprecise, informal keyword queries for searching over data. At the same time, the keyword search in some cases can be extremely slow. So further research on this field could make search even faster, with specific client and server side approaches. Moreover, an important future effort will be the consideration of extending or replacing system functionalities based on browsing with other client side databases (indexedDB, etc.). The usage of other client side databases, matured over time, can be used as a caching mechanism presenting useful browsing capabilities.

Finally, we can further research on prefetching techniques that allows to gather data that is potentially useful for subsequent queries based on semantic information derived from past queries based on semantic information. These prefetching techniques could also elevate the browsing experience.

Appendix A

Appendix

A.1 Keyword Search Queries

Classes - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf−schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?class
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}UNION{
        ?class rdf:type owl:Class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}UNION{
        ?class rdf:type rdfs:Class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}
FILTER (
        ?class!=owl:FunctionalProperty &&
        ?class!=owl:disjointWith&&
        ?class!=owl:AnnotationProperty &&
```

```
        ?class!=owl:InverseFunctionalProperty &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:SymmetricProperty &&
        ?class!=owl:DeprecatedClass &&
        ?class!=owl:DeprecatedProperty &&
        ?class!=owl:DataRange &&
        ?class!=owl:DatatypeProperty &&
        ?class!=owl:Ontology &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:Thing &&
        ?class!=owl:Restriction &&
        ?class!=owl:ObjectProperty &&
        ?class!=owl:Nothing &&
        ?class!=owl:AllDifferent &&
        ?class!=owl:NamedIndividual &&
        ?class!=owl:Class &&
        ?class!=owl:OntologyProperty &&
        ?class!=rdfs:Class &&
        ?class!=rdf:Property &&
        ?class!=rdf:List &&
        ?class!=rdfs:ContainerMembershipProperty &&
        ?class!=rdfs:Container &&
        ?class!=rdfs:Literal &&
        ?class!=rdfs:Datatype &&
        ?class!=rdfs:Resource &&
        ?class!=rdf:Statement &&
        ?class!=rdf:Alt &&
        ?class!=rdf:Seq &&
        ?class!=rdf:Bag &&
        ?class!=rdf:XMLLiteral &&
        !isBlank(?class)
        )
        FILTER regex(str(?class),'Keyword','i')
        FILTER regex(str(?label),'Keyword','i')
        FILTER regex(str(?comment),'Keyword','i')
}LIMIT 100
```

Classes - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
SELECT DISTINCT ?class
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}UNION{
        ?class rdf:type owl:Class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}UNION{
        ?class rdf:type rdfs:Class
        .OPTIONAL{?class rdfs:comment ?comment }
        .OPTIONAL{?class rdfs:label ?label }
}
FILTER (
    !STRSTARTS(STR(?class),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?class)
    )
FILTER regex(str(?class),'Keyword','i')
FILTER regex(str(?label),'Keyword','i')
FILTER regex(str(?comment),'Keyword','i')
}LIMIT 100
```

Properties - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
}UNION{
```

```
        ?prop rdf:type owl:ObjectProperty
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
}
FILTER (
        ?prop!=owl:sameAs &&
        ?prop!=owl:differentFrom &&
        ?prop!=owl:versionInfo &&
        ?prop!=owl:priorVersion &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:oneOf &&
        ?prop!=owl:unionOf &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:hasValue &&
        ?prop!=owl:disjointWith &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:allValuesFrom &&
        ?prop!=owl:cardinality &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:distinctMembers &&
        ?prop!=owl:equivalentClass &&
        ?prop!=owl:equivalentProperty &&
        ?prop!=owl:imports &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:intersectionOf &&
        ?prop!=owl:inverseOf &&
        ?prop!=owl:maxCardinality &&
        ?prop!=owl:minCardinality &&
        ?prop!=owl:onProperty &&
        ?prop!=owl:someValuesFrom &&
        ?prop!=rdfs:member &&
        ?prop!=rdfs:range &&
        ?prop!=rdfs:domain &&
        ?prop!=rdf:type &&
```

```
        ?prop!=rdfs:subClassOf &&
        ?prop!=rdfs:subPropertyOf &&
        ?prop!=rdfs:label &&
        ?prop!=rdfs:comment &&
        ?prop!=rdf:subject &&
        ?prop!=rdf:predicate &&
        ?prop!=rdf:object &&
        ?prop!=rdfs:seeAlso &&
        ?prop!=rdfs:isDefinedBy &&
        ?prop!=rdf:first &&
        ?prop!=rdf:rest &&
        ?prop!=rdf:nil &&
        ?prop!=rdf:value
        )
        FILTER regex(str(?prop),'Keyword','i')
        FILTER regex(str(?label),'Keyword','i')
        FILTER regex(str(?comment),'Keyword','i')
} LIMIT 100
```

Properties - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
}UNION{
        ?prop rdf:type owl:ObjectProperty
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
        .OPTIONAL{?prop rdfs:comment ?comment }
        .OPTIONAL{?prop rdfs:label ?label }
```

```
}
FILTER (
    !STRSTARTS(STR(?prop),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/1999/02/22-rdf-syntax-ns')
    )
FILTER regex(str(?prop),'Keyword','i')
FILTER regex(str(?label),'Keyword','i')
FILTER regex(str(?comment),'Keyword','i')
}LIMIT 100
```

Individuals - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?uri
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
        .OPTIONAL{?uri rdfs:comment ?comment }
        .OPTIONAL{?uri rdfs:label ?label }
}
UNION{
        ?uri rdf:type owl:NamedIndividual
        .OPTIONAL{?uri rdfs:comment ?comment }
        .OPTIONAL{?uri rdfs:label ?label }
}
FILTER (
        ?y!=owl:FunctionalProperty &&
        ?y!=owl:AnnotationProperty &&
        ?y!=owl:InverseFunctionalProperty &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:SymmetricProperty &&
        ?y!=owl:DeprecatedClass &&
        ?y!=owl:DeprecatedProperty &&
        ?y!=owl:DataRange &&
        ?y!=owl:DatatypeProperty &&
        ?y!=owl:Ontology &&
        ?y!=owl:TransitiveProperty &&
```

```
        ?y!=owl:Thing &&
        ?y!=owl:Restriction &&
        ?y!=owl:ObjectProperty &&
        ?y!=owl:Nothing &&
        ?y!=owl:AllDifferent &&
        ?y!=owl:NamedIndividual &&
        ?y!=owl:Class &&
        ?y!=owl:OntologyProperty&&
        ?y!=rdfs:Class &&
        ?y!=rdf:Property &&
        ?y!=rdf:List &&
        ?y!=rdf:Alt &&
        ?y!=rdf:Seq &&
        ?y!=rdf:Bag &&
        ?y!=rdfs:ContainerMembershipProperty &&
        ?y!=rdfs:Container &&
        ?y!=rdfs:Literal &&
        ?y!=rdfs:Datatype &&
        ?y!=rdfs:Resource &&
        ?y!=rdf:Statement &&
        ?y!=rdf:XMLLiteral &&
        !isBlank(?uri)
        )
        FILTER regex(str(?uri),'Keyword','i')
        FILTER regex(str(?label),'Keyword','i')
        FILTER regex(str(?comment),'Keyword','i')
}LIMIT 100
```

Individuals - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?uri
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
        .OPTIONAL{?uri rdfs:comment ?comment }
        .OPTIONAL{?uri rdfs:label ?label }
}
```

```
UNION{
        ?uri rdf:type owl:NamedIndividual
        .OPTIONAL{?uri rdfs:comment ?comment }
        .OPTIONAL{?uri rdfs:label ?label }
}
FILTER (
    !STRSTARTS(STR(?y),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?uri)
    )
FILTER regex(str(?uri),'Keyword','i')
FILTER regex(str(?label),'Keyword','i')
FILTER regex(str(?comment),'Keyword','i')
} LIMIT 100
```

A.2 ASK Queries

Classes - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
        ?class!=owl:FunctionalProperty &&
        ?class!=owl:disjointWith&&
        ?class!=owl:AnnotationProperty &&
        ?class!=owl:InverseFunctionalProperty &&
        ?class!=owl:TransitiveProperty &&
```

```
        ?class!=owl:SymmetricProperty &&
        ?class!=owl:DeprecatedClass &&
        ?class!=owl:DeprecatedProperty &&
        ?class!=owl:DataRange &&
        ?class!=owl:DatatypeProperty &&
        ?class!=owl:Ontology &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:Thing &&
        ?class!=owl:Restriction &&
        ?class!=owl:ObjectProperty &&
        ?class!=owl:Nothing &&
        ?class!=owl:AllDifferent &&
        ?class!=owl:NamedIndividual &&
        ?class!=owl:Class &&
        ?class!=owl:OntologyProperty &&
        ?class!=rdfs:Class &&
        ?class!=rdf:Property &&
        ?class!=rdf:List &&
        ?class!=rdfs:ContainerMembershipProperty &&
        ?class!=rdfs:Container &&
        ?class!=rdfs:Literal &&
        ?class!=rdfs:Datatype &&
        ?class!=rdfs:Resource &&
        ?class!=rdf:Statement &&
        ?class!=rdf:Alt &&
        ?class!=rdf:Seq &&
        ?class!=rdf:Bag &&
        ?class!=rdf:XMLLiteral &&
        !isBlank(?class)
        )
        FILTER(?class = <Resource URI>)
}
```

Classes - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
FROM <Graph name if selected>
WHERE {
```

```
{
        [] rdf:type ?class
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
    !STRSTARTS(STR(?class),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?class)
    )
FILTER(?class = <Resource URI>)
}
```

Properties - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
        ?prop!=owl:sameAs &&
        ?prop!=owl:differentFrom &&
        ?prop!=owl:versionInfo &&
        ?prop!=owl:priorVersion &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:oneOf &&
```

```
        ?prop!=owl:unionOf &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:hasValue &&
        ?prop!=owl:disjointWith &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:allValuesFrom &&
        ?prop!=owl:cardinality &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:distinctMembers &&
        ?prop!=owl:equivalentClass &&
        ?prop!=owl:equivalentProperty &&
        ?prop!=owl:imports &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:intersectionOf &&
        ?prop!=owl:inverseOf &&
        ?prop!=owl:maxCardinality &&
        ?prop!=owl:minCardinality &&
        ?prop!=owl:onProperty &&
        ?prop!=owl:someValuesFrom &&
        ?prop!=rdfs:member &&
        ?prop!=rdfs:range &&
        ?prop!=rdfs:domain &&
        ?prop!=rdf:type &&
        ?prop!=rdfs:subClassOf &&
        ?prop!=rdfs:subPropertyOf &&
        ?prop!=rdfs:label &&
        ?prop!=rdfs:comment &&
        ?prop!=rdf:subject &&
        ?prop!=rdf:predicate &&
        ?prop!=rdf:object &&
        ?prop!=rdfs:seeAlso &&
        ?prop!=rdfs:isDefinedBy &&
        ?prop!=rdf:first &&
        ?prop!=rdf:rest &&
        ?prop!=rdf:nil &&
        ?prop!=rdf:value
        )
        FILTER(?prop = <Resource URI>)
}
```

Properties - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
    !STRSTARTS(STR(?prop),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/1999/02/22-rdf-syntax-ns')
    )
FILTER(?prop = <Resource URI>)
}
```

Individuals - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
}
UNION{
        ?uri rdf:type owl:NamedIndividual
}
FILTER (
        ?y!=owl:FunctionalProperty &&
        ?y!=owl:AnnotationProperty &&
```

```
        ?y!=owl:InverseFunctionalProperty &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:SymmetricProperty &&
        ?y!=owl:DeprecatedClass &&
        ?y!=owl:DeprecatedProperty &&
        ?y!=owl:DataRange &&
        ?y!=owl:DatatypeProperty &&
        ?y!=owl:Ontology &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:Thing &&
        ?y!=owl:Restriction &&
        ?y!=owl:ObjectProperty &&
        ?y!=owl:Nothing &&
        ?y!=owl:AllDifferent &&
        ?y!=owl:NamedIndividual &&
        ?y!=owl:Class &&
        ?y!=owl:OntologyProperty&&
        ?y!=rdfs:Class &&
        ?y!=rdf:Property &&
        ?y!=rdf:List &&
        ?y!=rdf:Alt &&
        ?y!=rdf:Seq &&
        ?y!=rdf:Bag &&
        ?y!=rdfs:ContainerMembershipProperty &&
        ?y!=rdfs:Container &&
        ?y!=rdfs:Literal &&
        ?y!=rdfs:Datatype &&
        ?y!=rdfs:Resource &&
        ?y!=rdf:Statement &&
        ?y!=rdf:XMLLiteral &&
        !isBlank(?uri)
        )
        FILTER(?uri = <Resource URI>)
}
```

Individuals - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

ASK
```

```
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
}
UNION{
        ?uri rdf:type owl:NamedIndividual
}
FILTER (
    !STRSTARTS(STR(?y),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?uri)
    )
FILTER(?uri = <Resource URI>)
}
```

A.3 Count Resources Queries

Classes - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT COUNT(DISTINCT ?class)
FROM <Graph name if selected>
WHERE {
{
        [] rdf:type ?class
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
        ?class!=owl:FunctionalProperty &&
        ?class!=owl:disjointWith&&
        ?class!=owl:AnnotationProperty &&
        ?class!=owl:InverseFunctionalProperty &&
```

```
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:SymmetricProperty &&
        ?class!=owl:DeprecatedClass &&
        ?class!=owl:DeprecatedProperty &&
        ?class!=owl:DataRange &&
        ?class!=owl:DatatypeProperty &&
        ?class!=owl:Ontology &&
        ?class!=owl:TransitiveProperty &&
        ?class!=owl:Thing &&
        ?class!=owl:Restriction &&
        ?class!=owl:ObjectProperty &&
        ?class!=owl:Nothing &&
        ?class!=owl:AllDifferent &&
        ?class!=owl:NamedIndividual &&
        ?class!=owl:Class &&
        ?class!=owl:OntologyProperty &&
        ?class!=rdfs:Class &&
        ?class!=rdf:Property &&
        ?class!=rdf:List &&
        ?class!=rdfs:ContainerMembershipProperty &&
        ?class!=rdfs:Container &&
        ?class!=rdfs:Literal &&
        ?class!=rdfs:Datatype &&
        ?class!=rdfs:Resource &&
        ?class!=rdf:Statement &&
        ?class!=rdf:Alt &&
        ?class!=rdf:Seq &&
        ?class!=rdf:Bag &&
        ?class!=rdf:XMLLiteral &&
        !isBlank(?class)
)}
```

Classes - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT COUNT(DISTINCT ?class)
FROM <Graph name if selected>
WHERE {
{
```

```
        [] rdf:type ?class
}UNION{
        ?class rdf:type owl:Class
}UNION{
        ?class rdf:type rdfs:Class
}
FILTER (
    !STRSTARTS(STR(?class),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?class),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?class)
    )
}
```

Properties - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT count(DISTINCT) ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
        ?prop!=owl:sameAs &&
        ?prop!=owl:differentFrom &&
        ?prop!=owl:versionInfo &&
        ?prop!=owl:priorVersion &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:oneOf &&
        ?prop!=owl:unionOf &&
        ?prop!=owl:complementOf &&
```

```
        ?prop!=owl:hasValue &&
        ?prop!=owl:disjointWith &&
        ?prop!=owl:backwardCompatibleWith &&
        ?prop!=owl:allValuesFrom &&
        ?prop!=owl:cardinality &&
        ?prop!=owl:complementOf &&
        ?prop!=owl:distinctMembers &&
        ?prop!=owl:equivalentClass &&
        ?prop!=owl:equivalentProperty &&
        ?prop!=owl:imports &&
        ?prop!=owl:incompatibleWith &&
        ?prop!=owl:intersectionOf &&
        ?prop!=owl:inverseOf &&
        ?prop!=owl:maxCardinality &&
        ?prop!=owl:minCardinality &&
        ?prop!=owl:onProperty &&
        ?prop!=owl:someValuesFrom &&
        ?prop!=rdfs:member &&
        ?prop!=rdfs:range &&
        ?prop!=rdfs:domain &&
        ?prop!=rdf:type &&
        ?prop!=rdfs:subClassOf &&
        ?prop!=rdfs:subPropertyOf &&
        ?prop!=rdfs:label &&
        ?prop!=rdfs:comment &&
        ?prop!=rdf:subject &&
        ?prop!=rdf:predicate &&
        ?prop!=rdf:object &&
        ?prop!=rdfs:seeAlso &&
        ?prop!=rdfs:isDefinedBy &&
        ?prop!=rdf:first &&
        ?prop!=rdf:rest &&
        ?prop!=rdf:nil &&
        ?prop!=rdf:value
)}
```

Properties - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
SELECT count(DISTINCT) ?prop
FROM <Graph name if selected>
WHERE {
{
        ?prop rdf:type rdf:Property
}UNION{
        ?prop rdf:type owl:ObjectProperty
}UNION {
        [] ?prop ?o
}UNION {
        ?prop rdf:type owl:DatatypeProperty
}
FILTER (
    !STRSTARTS(STR(?prop),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?prop),'http://www.w3.org/1999/02/22-rdf-syntax-ns')
    )
}
```

Individuals - SPARQL 1.0

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT count(DISTINCT ?uri)
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
}
UNION{
        ?uri rdf:type owl:NamedIndividual
}
FILTER (
        ?y!=owl:FunctionalProperty &&
        ?y!=owl:AnnotationProperty &&
        ?y!=owl:InverseFunctionalProperty &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:SymmetricProperty &&
        ?y!=owl:DeprecatedClass &&
        ?y!=owl:DeprecatedProperty &&
```

```
        ?y!=owl:DataRange &&
        ?y!=owl:DatatypeProperty &&
        ?y!=owl:Ontology &&
        ?y!=owl:TransitiveProperty &&
        ?y!=owl:Thing &&
        ?y!=owl:Restriction &&
        ?y!=owl:ObjectProperty &&
        ?y!=owl:Nothing &&
        ?y!=owl:AllDifferent &&
        ?y!=owl:NamedIndividual &&
        ?y!=owl:Class &&
        ?y!=owl:OntologyProperty&&
        ?y!=rdfs:Class &&
        ?y!=rdf:Property &&
        ?y!=rdf:List &&
        ?y!=rdf:Alt &&
        ?y!=rdf:Seq &&
        ?y!=rdf:Bag &&
        ?y!=rdfs:ContainerMembershipProperty &&
        ?y!=rdfs:Container &&
        ?y!=rdfs:Literal &&
        ?y!=rdfs:Datatype &&
        ?y!=rdfs:Resource &&
        ?y!=rdf:Statement &&
        ?y!=rdf:XMLLiteral &&
        !isBlank(?uri)
)}
```

Individuals - SPARQL 1.1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT count(DISTINCT ?uri)
FROM <Graph name if selected>
WHERE {
{
        ?uri rdf:type ?y
}
UNION{
        ?uri rdf:type owl:NamedIndividual
```

```
}
FILTER (
    !STRSTARTS(STR(?y),'http://www.w3.org/2002/07/owl')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/2000/01/rdf-schema')&&
    !STRSTARTS(STR(?y),'http://www.w3.org/1999/02/22-rdf-syntax-ns')&&
    !isBlank(?uri)
    )
}
```

Blank Nodes

```
SELECT count(distinct ?x)
FROM <Graph name if selected>
WHERE {
        ?x ?y ?z
        FILTER(isBlank(?x))
}
```

A.4 Labels and abstracts

Without filtering

```
SELECT ?label ?comment
FROM <Graph name if selected>
WHERE {
  {
        <Resource URI> rdfs:label ?label
  }
  UNION{
        <Resource URI> rdfs:comment ?comment
  }
} LIMIT 2
```

With filtering

```
SELECT ?label ?comment
FROM <Graph name if selected>
WHERE {
  {
        <Resource URI> rdfs:label ?label
        FILTER(lang(?label)='SELECTED LANGUAGE CODE' ||
        lang(?label)='')
  }
```

```
  UNION{
        <Resource URI> rdfs:comment ?comment
        FILTER(lang(?comment)='SELECTED LANGUAGE CODE' ||
        lang(?comment)='')
  }
} LIMIT 2
```

Bibliography

[1] M. Lawson, "Berners-lee on the read/write web," *BBC News*, vol. 9, 2005.

[2] R. Berjon, S. Faulkner, T. Leithead, S. Pfeiffer, E. O'Connor, and E. D. Navara, "HTML5," W3C, Candidate Recommendation, Jul. 2014, http://www.w3.org/TR/2014/CR-html5-20140731/.

[3] R. Guha and D. Brickley, "RDF schema 1.1," W3C, W3C Recommendation, Feb. 2014, http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[4] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, W3C Recommendation, Jan. 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[5] I. Hickson, "Web SQL database," W3C, W3C Note, Nov. 2010, http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/.

[6] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification," 1999.

[7] G. Klyne and J. J. Carroll, "Resource description framework (rdf): Concepts and abstract syntax," 2006.

[8] T. Gruber, I. L. L. Ontology, and M. T. Özsu, "Encyclopedia of database systems," *Liu & T. Ozsu, eds. Encyclopedia of Database Systems*, 2008.

[9] M. Laine, "Client-side storage in web applications," Aalto University, Technical Report, Tech. Rep., 2012.

[10] I. Hickson, "Web storage," W3C, W3C Recommendation, Jul. 2013, http://www.w3.org/TR/2013/REC-webstorage-20130730/.

[11] J. Orlow, J. Bell, N. Mehta, A. Popescu, J. Sicking, and E. Graff, "Indexed database API," W3C, Candidate Recommendation, Jul. 2013, http://www.w3.org/TR/2013/CR-IndexedDB-20130704/.

[12] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets, "Tabulator: Exploring and analyzing linked data on the semantic web," in *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, vol. 2006. Athens, Georgia, 2006.

[13] C. Bizer and T. Gauß, "Disco-hyperdata browser: A simple browser for navigating the semantic web," 2007.

[14] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs, "Swoogle: a search and metadata engine for the semantic web," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management.* ACM, 2004, pp. 652–659.

[15] L. Rietveld and R. Hoekstra, "Yasgui: Not just another sparql client," in *The Semantic Web: ESWC 2013 Satellite Events.* Springer, 2013, pp. 78–86.

[16] K. Elbedweihy, S. Mazumdar, S. N. Wrigley, and F. Ciravegna, "Nl-graphs: A hybrid approach toward interactively querying semantic data," in *The Semantic Web: Trends and Challenges.* Springer, 2014, pp. 565–579.

[17] M. Papadakis and Y. Tzitzikas, "Answering keyword queries through cached subqueries in best match retrieval models," *Journal of Intelligent Information Systems*, vol. 44, no. 1, pp. 67–106, 2015.

[18] N. Manolis and Y. Tzitzikas, "Interactive exploration of fuzzy rdf knowledge bases," in *The Semantic Web: Research and Applications.* Springer, 2011, pp. 1–16.

[19] G. T. Williams and J. Weaver, "Enabling fine-grained http caching of sparql query results," in *The Semantic Web–ISWC 2011.* Springer, 2011, pp. 762–777.

[20] M. Martin, J. Unbehauen, and S. Auer, "Improving the performance of semantic web applications with sparql query caching," in *The Semantic Web: Research and Applications.* Springer, 2010, pp. 304–318.

[21] S. Ferré, "Expressive and scalable query-based faceted search over sparql endpoints," in *The Semantic Web–ISWC 2014*. Springer, 2014, pp. 438–453.

[22] M. Janevska, M. Jovanovik, and D. Trajanov, "Html5 based facet browser for sparql endpoints."

[23] J. Lorey and F. Naumann, "Caching and prefetching strategies for sparql queries," in *The Semantic Web: ESWC 2013 Satellite Events*. Springer, 2013, pp. 46–65.

[24] W. Fan, X. Wang, and Y. Wu, "Answering graph pattern queries using views," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 184–195.

[25] R. Cyganiak, J. Zhao, M. Hausenblas, and K. Alexander, "Describing linked datasets with the VoID vocabulary," W3C, W3C Note, Mar. 2011, http://www.w3.org/TR/2011/NOTE-void-20110303/.

[26] R. Blanco, P. Mika, and S. Vigna, "Effective and efficient entity search in rdf data," in *The Semantic Web–ISWC 2011*. Springer, 2011, pp. 83–97.

[27] S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum, "Searching rdf graphs with sparql and keywords." *IEEE Data Eng. Bull.*, vol. 33, no. 1, pp. 16–24, 2010.

[28] T. Tran, H. Wang, S. Rudolph, and P. Cimiano, "Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009, pp. 405–416.