

CHALMERS



Interactive construction of SPARQL queries using a schema-based graphical interface

*Master's Thesis in Computer Science Algorithms, Languages and
Logic*

IOANNIS SOTIRIADIS

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2012
Master's Thesis 2012

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Interactive construction of SPARQL queries using a schema-based graphical interface

IOANNIS SOTIRIADIS

©IOANNIS SOTIRIADIS, December 2012.

Examiner: GRAHAM KEMP

Chalmers University of Technology University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)72 022 8679

Department of Computer Science and Engineering
Gothenburg, Sweden December 2012

Abstract

This Master thesis presents a visual querying interface, the Visual Navigator (VN), implemented for Semantic Web. VN is a tool that takes an RDF schema and creates an interactive Entity-Relationship diagram. The user of the tool can interact with the E-R diagram and create SPARQL queries incrementally. After creating the query, the user can submit it and get the results. The results are presented in a table in which result values can be highlighted to automatically add and refine selection criteria in follow-on queries. The resulting system offers a new way to query the Semantic Web which is different to RDF browsers or query interfaces that are based on graphs of RDF triples.

Acknowledgements

I would like to thank P.M.D. Gray for his help and for offering us the source code of Visual Navigator and of course G.J.L. Kemp for his help and guidance through the whole period of time.

Ioannis Sotiriadis, Gothenburg December 2012

Contents

1	Introduction	1
1.1	The Problem	1
1.1.1	Entity-Relationship model	1
1.1.2	Semantic Web	2
1.1.3	Graphical Interface for constructing queries	3
1.2	The Scope of this thesis	3
2	Background	5
2.1	Function Data Model	5
2.1.1	Daplex a query language for FDM	6
2.1.1.1	Daplex construction statements	6
2.1.1.2	Daplex query statements and expressions	7
2.1.2	Visual Navigator for Daplex	8
2.1.2.1	Entity Editor	8
2.1.2.2	Expression Editor	9
2.1.2.3	Response Window	10
2.2	Resource Description Framework	11
2.2.1	The Data Model	11
2.2.2	SPARQL	13
3	Visual Navigator for SPARQL	17
3.1	Implementation	17
3.1.1	Query GUI	17
3.1.2	Entity Editor	18
3.1.3	Result Window	20
3.1.4	Examples	20
3.2	Limitations	22

4	Related Work	25
4.1	Browsers creating query language queries	25
4.2	Browsers for graph queries	26
4.3	Other kind of applications	27
5	Discussion and Future Work	28
A	Manual	30
B	Implementation	35
	Bibliography	39

1

Introduction

In today's society the amount of data in the World Wide Web (WWW) is extremely big and it keeps growing. Scientists and researchers are trying to come up with techniques, languages, standards etc. to be able to organize, represent and manipulate those data. This field has become a point of research and in the latter years there have been some promising results.

1.1 The Problem

As we said, since WWW and the amount of data is enlarging rapidly, the need to represent them and to be able to manipulate them was born. In 1976 Peter P. Chen [1] presented the Entity-Relationship (E-R) model. An E-R model is a model which represents data in an abstract way. This helps the users to understand easily the schema and in general the structure and the relationships of a database. Also in the last decade a new way of representing the data has been presented, the Semantic Web. Semantic Web helps the machines to process the data that are stored in the WWW. This gives endless opportunities which can be utilized. We are going to present the basic concepts of those two subjects in the rest of this chapter.

1.1.1 Entity-Relationship model

The E-R model consists of 3 parts the Entity sets, the Relationship sets and the Value sets. According to Merriam-Webster dictionary an entity is something that has separate and distinct existence and objective or conceptual reality. The same stands for the entities in the E-R model, but sometimes it may be the case that in the E-R model some entities will be sub-classes of another entity. For example it may be the case that we have an entity person and an entity teacher. Teacher is a person and will have all the properties that a person has but it may have some more specific properties for itself. Relationships are the connections between two or more entities. The relationships may

be one of the following three types: one to one (1:1), one to many(1:n) and many to many(m:n). Those three types represent the connection between the entities, for example in a school's E-R model a many to many relationship may be that a teacher (entity) teaches (relationship) many courses(entity), but also a course(entity) is been taught (same as teaches relationship) by many teachers(entity). Finally the values/attributes are the properties that are assigned to the entities. For example a teacher can have a first-name, last-name etc. In the next two figures we see the common representation of the many to many relationship between teacher and course in the classical way of E-R diagram representation(Figure 1.1) and a most recent representation which was build with the tool VisualParadigm¹ (Figure 1.2).

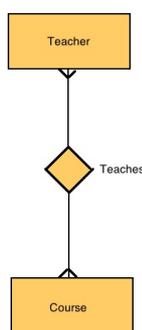


Figure 1.1: A classical representation of E-R model

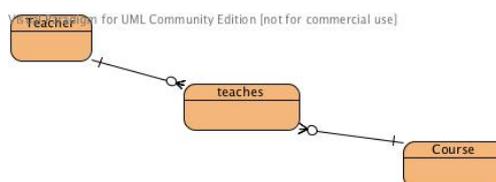


Figure 1.2: Visual Paradigm's E-R diagram

The visual representation of the E-R model is the E-R diagram. "Each entity set is represented by a rectangular box, and each relationship set is represented by a diamond-shaped box. " [1]. The relationship definitions on the entity sets are represented by lines connecting the rectangular boxes and the diamond-shaped boxes.

1.1.2 Semantic Web

In the early 2000's the idea of Semantic Web was presented by the inventor of WWW. The term of Semantic Web is presented in [2]. Semantic Web is an effort to make the content in the WWW accessible and readable for a machine. This means that the structure of the web pages should change and also that there should be a lot of information in metadata. At present, there are two technologies that help to do that : eXtensible Markup Language (XML) and Resource Description Framework (RDF) [2]. XML allows more freedom to the user, it allows him/her to make his/her own tags. Those tags can

¹<http://www.visual-paradigm.com/>

be recognized by the agents but they should know what is the purpose of every tag. "In short, XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean" [2]. The RDF is a more structured technique which can also be expressed through XML but it has standards. We will present RDF and RDF Schema (RDFS) more extensively in the following chapters.

The usage of Semantic Web according to W3C is "to provide a common framework that allows data to be shared and reused across application, enterprise, and community boundaries." [3]. The transition from the WWW to the Semantic Web will make a great difference in our everyday lives. But this transition is not simple, and that is what Nigel Shadbolt et. al realized in 2006 [4] where they clearly stated: "This simple idea, however, remains largely unrealized". Nowadays, in 2012, it seems that there has been some developments and some have started to believe that 2012 is the year of Semantic Web [5]. One great example of how our lives will change is the Siri application of Apple's iPhone 4S. Siri is a simpler form of the agents that T. Berners-Lee et al. had in mind in 2001 [2]. You can talk to your phone asking for info, for example someone could say "I would like to eat at a french restaurant", then Siri will give you a result of French restaurants that are in your area and it would make you a suggestion as well. Siri is also able to make reservation to the restaurant of your choice. And this is only one example on what Siri can do.

1.1.3 Graphical Interface for constructing queries

The new technologies are rapidly evolving, and the need of processing the vast amount of data is urgent even for the most inexperienced users. Since not everyone is able to manipulate and use the query languages efficiently there is a need to build applications to help them. One good way to help is to build graphical interfaces for constructing queries. This will help the users, even those that don't know anything about querying languages, to build correct queries. It would be also easy to do so, because the only thing that the user will have to do is to interact with the interface to show what he/she wants to retrieve from the query and then the application will automatically create the query. Of course if the user is an experienced one he/she can interact with the resulting query and modify it to get exactly the wanted results. With that we mean that the resulting queries may not be fully expressive and this is quite understandable because if we want to keep it simple in the construction for the user, we cannot construct complex queries.

1.2 The Scope of this thesis

In this thesis we will try to make a graphical interface for constructing queries in SPARQL, a query language for RDFS. Our application will be based on Visual Navigator (VN). VN is an application that helps the user to build Daplex queries for Functional Data Model (FDM) schemas. VN loads the schema and then it represents it as a 2-D Entity-Relationship diagram. The user can interact with the E-R diagram and build-up the query by clicking the entities and the relationships. Then the user can submit the

query to the database via a CORBA interface. The results are returned to a different window. There is also a "novel feature" [6] the "copy and drop" feature, which allows the user to redefine the query by filtering, using the results of the initial query.

The main purpose of the thesis is to change the implementation of VN and instead of helping the user to create Daplex queries, VN will help the user to create SPARQL queries. Those two query languages have similarities but also some major differences one of which is that they are using different data models. In Chapter 2, we will give some information about the background and more specifically about FDM, the initial VN and RDFS. In Chapter 3 we will present the current Visual Navigator and what are the changes that were made, and how those affected the initial application. In Chapter 4 we will see some of the related work that has been done in the field and finally there will be the conclusions and a discussion on what extent of SPARQL is covered and how the application can be extended or improved in the future.

2

Background

In this chapter we are going to present some basic concepts that are needed for the reader to understand easier the rest of this report. As well there will be reference to and a description of, an earlier work that have been done in the area of application that automatically construct queries. We will discuss two different ways of modeling data, Functional Data Model (FDM) and RDF Schema (RDFS). That is because our project is based on an earlier work, and there are similarities in both techniques, so it is important to present them so the reader will be able to understand how this project was based on a previous application, which made for a completely different schema and query language. And of course it makes it easier for the reader to see why we made the changes that we made at the application.

2.1 Function Data Model

A data model is a way of organizing data and databases. Normally the data and the data structure are separated. The Functional Data Model (FDM) [7] proposes a uniform representation of data and procedures. Its propose is that everything in the data model should be structured as a function. "The notion of a functional data model was first introduced by Sibley and Kershberg. This work explored the use of the functional approach as a tool for modeling the data structures representable under the three dominant data models (hierarchical, relational, and network)" [8]. There are three categories of functions:

- the database functions, which are all the functions that represent data objects, attributes and relationships,
- the user defined functions which are the Lambda Calculus that the user can define on his/her own,

- the data manipulation functions or else a Functional Data Manipulation Language, which are functions that can be applied to database functions.

2.1.1 Daplex a query language for FDM

One of the most known querying languages for FDM, and the one we are going to present, is Daplex. "Daplex is a functional-style data declaration and manipulation for functional data model databases" [9]. What Daplex tries to accomplish is to be a query language that resembles the human way of thinking when trying to solve a problem, set a query. This makes it easier for the user to construct queries because the connection between the mental and the formal representation of the query is direct [8]. Since Daplex is a language for function data the main constructors of it are the entities and the functions which help to model conceptual objects and their properties.

It is really hard to model data from real world. That is because for different people there are different things that are considered as entities or functions so it can be a conflict on how to do it and understand it. As well, some entities might have properties from other entities. Normally in the real life world each entity will have all of its properties, but in Daplex an entity might not have all of its properties shown but it is necessary to show that some properties are derived from other entities.

"In short, the DAPLEX language is an attempt to provide a database system interface which allows the user to more directly model the way he thinks about the problems he is trying to solve." [8]

In the rest of the section we are going to present Daplex and its properties as defined by Shipman in [8]

2.1.1.1 Daplex construction statements

The first point that are we are going to present here is how functions are constructed. "Functions are used to express both entity types and what we have been calling the properties of an entity" [8]. The statement that we are going to talk about first is the "Declare" statement, and we are going to do that with an example. Let us consider that we have the following statement :

```
declare age(person) -> Integer
```

Now, what does this mean? This means that "age" is a function which maps entities of type "person" to entities of type Integer. Integer along with others like String and Boolean are entity types provided by the system. This was one way of how someone can construct a declare statement, but there are some others as well. Now let us consider another statement:

```
declare supervises(teacher) -> > project
```

Here we see that the function "supervises" applied to the entity "teacher" returns a set of entities of type "project". The double headed arrow indicates that "supervises" is a multivalued function. This means that a teacher may supervise more than one project. The returning set in that kind of functions may be even the empty set. There are also single valued functions.

```
declare birthday(person) -> date
```

This statement now shows the function "date" applied on an entity "person" returns an entity of type "date". The single headed arrow and the fact that the return is a single entity as well qualifies this function as a single valued function. And finally we have the declaration of an entity:

```
declare person -> > entity
```

This shows that the function person returns a set of entities. "ENTITY is the system-provided type of all entities" [8]. Sometimes, though, instead of declaring an entity someone may want to define a function that will return an entity. For example a teacher entity may be defined to be a person entity. This can be done with the "Define" statement. "Essentially we are defining new properties of objects based on the values of other properties." [8].

2.1.1.2 Daplex query statements and expressions

To build a simple Daplex query the user should rely on three basic statements the "For Each" statement, the "Such That" statement and the "Print" statement. According to Shipman[8] the three statements work as follow:

- The "For Each" statement iterates over the given set of entities.
- The "Such That" statement accompanies the "For Each" statement and it is a kind of a filter. What it does is that it tests all the entities of the given set in the "For Each" statement against a qualifying predicate, an expression.
- The "Print" statement executes the given set of entities' for-body for each member of the set.

Besides those three basic statements there are more statements that add to the expressiveness of Daplex. Some of those statements we are going to describe now.

- The "For Some" statement is an extra qualification that goes along with the "Such That" statement and with that the query refers to a particular entity from the set that is considered from "Such That".
- The functions "Average", "Total", "Count", "Maximum" and "Minimum" are functions that are applied after the "Print" statement. We will use here the example that Shipman [8] uses consider the query print count (Instructor Such That Name (Dept(Instructor)) = "EE"). The argument to the "Count" function is a set of "Instructor" entities, and "Count" simply returns the cardinality of that set.
- The "Value" expression evaluates each expression and returns the set of entities of this evaluation.
- The "Role" expression defines the entity type that the interpreter should use when there is a need to resolve external function name ambiguities.
- The "Order" expression as its name implies is the ordering of the entities.

2.1.2 Visual Navigator for Daplex

Visual Navigator is a Java-based application, which was presented by I. Gil et al. [6]. It is a visual query system for databases which helps the user to automatically create Daplex queries for P/FDM. P/FDM is a Prolog implementation of the Functional Data Model [10]. Firstly the user has the main window (Figure 2.1) in which the user can select and load the P/FDM schema that he/she wants. There the application parses the schema and creates in Java the appropriate objects for entities, relationships and functions. It also reads from another file some coordinates (x,y) that it will match them with the created objects for entities and relationships. This match is needed because afterwards all of those objects are going to be represented in a 2-D E-R diagram.



Figure 2.1: Welcome Screen

2.1.2.1 Entity Editor

After loading the schema, there is an option for visually representing it. This visual representation is done in a new window, the Entity Editor window (Figure 2.2). The Entity Editor consists of two parts: the BoxCanvas, where the 2-D E-R diagram is drawn, and the QueryCanvas, where the user can see the resulting query. The BoxCanvas uses information from the earlier process of the schema and it makes a visual representation of the data in a 2-D E-R diagram. The entities are represented as ellipses and "the relationships are represented as arcs between entity classes and a "Crow's Feet" at the end of a relationship indicate a multi-valued relationship" [6]. The relationships have also a circle in the middle which help for the interaction. The functions are not showed in the E-R diagram, but they are joined with the entity classes that they belong to.

The E-R diagram is interactive. The user can add "for each" statements to the query for both entities and relationships. In addition, the user can change the position of the entities and relationships in the E-R diagram.

The user can also interact with the QueryCanvas. As stated earlier, QueryCanvas is where the user can see the resulting query from the actions that were made and check if it is what it was expected. If the query does not satisfy or represent what was intended to express, the user can interact with the QueryCanvas and change it. The way that this is done can differ. Entities and/or relationships can be deleted from the query. Attributes for the "print" statement can be added to the query, or removed from that. And finally there can be condition filters to the query by the addition of "such that" statements. To create the "such that" statements the user must use the Expression Editor window.

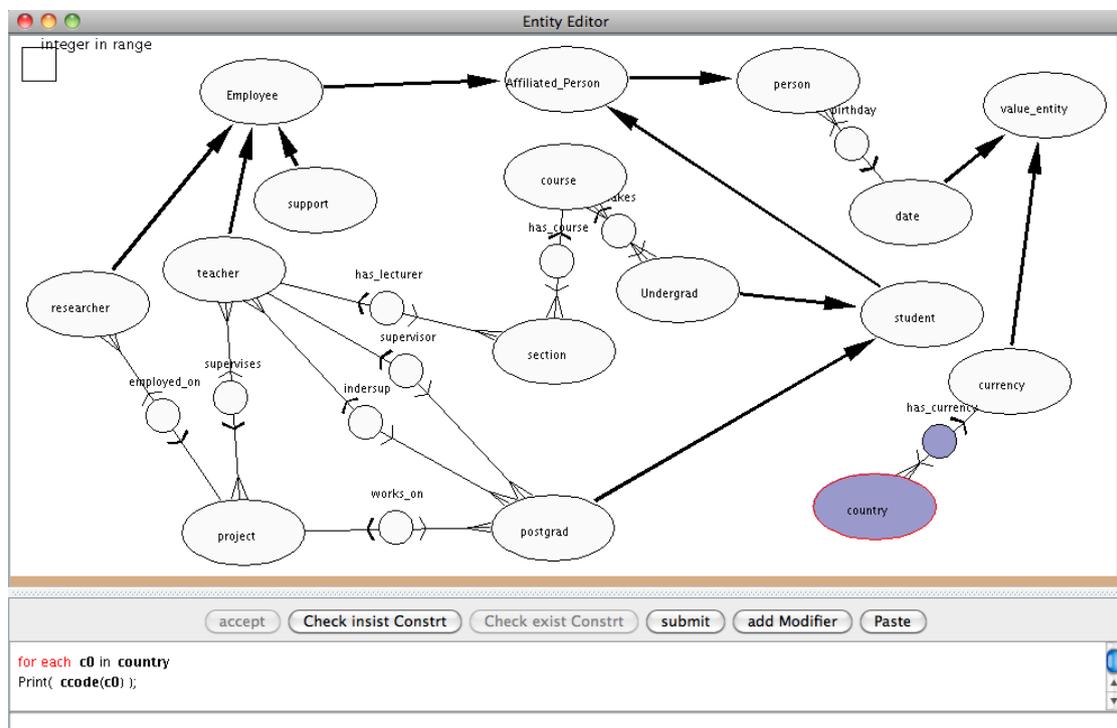


Figure 2.2: Entity Editor

2.1.2.2 Expression Editor

The Expression Editor (Figure 2.3) is a window where the user can create a condition filter for the query. The expressions in the Expression Editor are composed from three parts, the left hand-side term, the operator and the right hand-side term. The left hand-side can either be a constant or a function. "Select Function" brings up the Function Chooser window, selecting the function to be constrained and then typing a value into the entry box" [6]. In the right hand-side the user can select if what is going to be

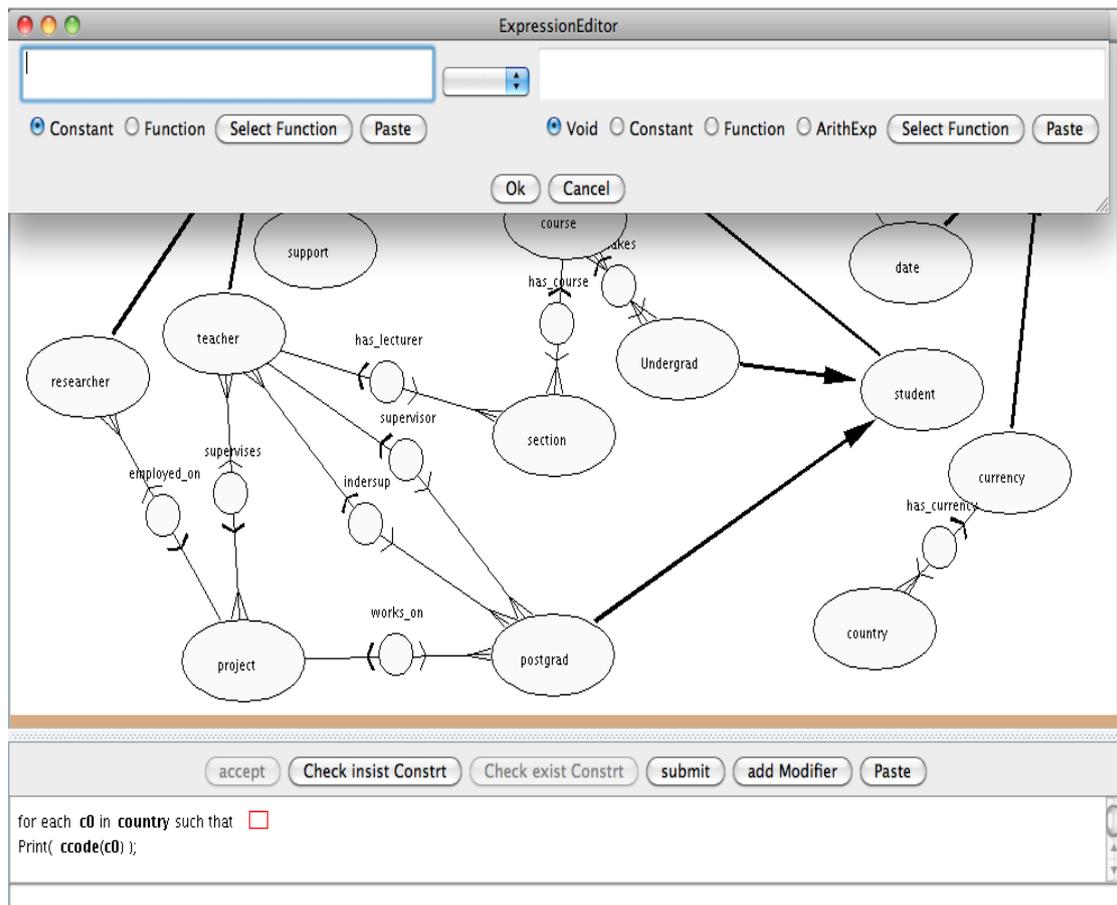


Figure 2.3: Expression Editor

constrained will have a void value, an arithmetic value or again if it is going to be a function. The operators are the usual comparison operators like "==" , ">" etc. Finally in the left side, if the user wants to have more than one constraints, after the first constraint it appears an operator which can be either an OR or an AND depending on what the user want to do, to form a conjunction or disjunction the new constraint with the rest.

2.1.2.3 Response Window

After the construction of the query the user can execute it. VN sends the query to the server and then retrieves the results. This connection is made in the background. After getting the first row from the results a new window, which will contain all the results, pops-up. This new window is the Response window. This window is composed from two parts. In the lower part there is panel which shows the originated query. On the top there is a table with the results. The table has labels for its columns. The names of the

columns are the functions/attributes the user selected to print statement in the query. The rows of the table contains the expected results.

The implementation of that table is made using the swing class `JTable`. To get the results from the server, though, there was a need to override and redefine some default methods. This was done by creating the class `VectorTableModel`. "The class `VectorTableModel` is an abstract class, the method `submit` has to be redefined to access the database and get the result. New lines are added with the method `addRow`" which has a parameter a vector of strings. It sends the MVC event to the `JTable` view." [6].

One other feature that exists in the Response Window is the "Copy-and-Drop" facility. With this facility the user can select cells from the result table and filter even more the query and create "such that" statements. The statement is created automatically. Depending on the values chosen from the user the expression can be joined with OR if the cells are in the same column or with AND if they are in the same row. The AND terms precede the OR terms, so if there are cells from the same row and the same cell the grouping will be all the cells from the same row in a parentheses joined with AND and then all the cells of the next row (again joined with AND) will be joined together with OR. When the user adds the statement, the statement is added automatically to the correct place.

2.2 Resource Description Framework

"RDF is a standard published by W3C, and it can be used to represent distributed information/knowledge in a way that computer applications can use and process in a scalable manner" [11]. In chapter 1 we saw that it is starting to becoming more and more important to automatize the extraction of information from the WWW because of the amount of the information, and it is quite hard for someone to do it manually. But as we saw, to do that needs a transition to Semantic Web, so that machines will be able to read the content of the sites.

2.2.1 The Data Model

We have discussed the reasons that we needed a new data model that can process meta-data and combine information, let's see now the goals of that model. According to W3C the goals for RDF are:

- have a simple data model
- have formal semantics and provable inference
- use an extensible URI-based vocabulary
- use an XML-based syntax
- supporting use of XML schema datatypes
- allow anyone to make statements about any resource [12].

We can say that RDF mainly focuses on 3 points [11]. One point is that every statement should be a triple of the type subject-predicate-object where the subject and the object are entities and the predicate is the relationship between them. The second point is that both the predicate and the resource must be global and URI based. And the last point is that anyone can make statements about any resource.

What W3C proposed in 2004 was RDF W3C Recommendation which is a join of six previous specifications about RDF. Those six documents are:

- RDF Primer
- RDF Test Cases
- RDF Concept
- RDF Semantics
- RDF Schema
- RDF Syntax [11].

The first point, we said, is that every statement should be a triple. That is the main and most important characteristic of RDF, and that helps for each piece of information to be clearly defined. A single statement represents a single fact, while a collection of statements represents information or knowledge. The collection of statements are also called RDF Graphs [11]. The second point is that every resource should be a global Uniform Resource Identifier (URI) or a blank node. URI, as its name implies, is an identifier that represents uniquely an entity or a relationship in the web. That means for example if someone wants to represent Ioannis Sotiriadis, in the web, then assuming that it will be in Chalmers domain, it would look something like that: `www.chalmers.se/person/student#Ioannis_Sotiriadis` or instead of a hash we can have a slash `www.chalmers.se/person/student/Ioannis_Sotiriadis`. That will represent the person as an entity and someone that will want to retrieve information about that person can use it. The slash(/) and the hash(#) are two different ways of constructing an URI, the most common though is the hash. A blank node is a node with no name, it doesn't have a URI as its identifier [11]. The reason why we want to represent everything with URIs is that we want to have uniformity and everyone that wants to add or retrieve information from a single entity then he/she should use the URI so it will be constant with the rest users of the world. This means that the users should search to find if a URI for the entity that they want to use already exists and retrieve it. It is of great importance that the users will use the existing URI and create their own as a last resort. The last point is pretty much obvious why it is a need that anyone can make statements about any resource. In RDF the values of a subject, predicate or object might be either RDF literal, which are the characters that are allowed to be used, or they might be a blank node.

There are two types of syntaxes for RDF, the first and most common one is the XML syntax and the second is Notation 3, which W3C introduced to make easier to write by

hand RDF triples. For the XML syntax, according to [13], the classes are shown in Table 2.1 and the properties in Table 2.2.

Class name	comment
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:XMLLiteral	The class of XML literal values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.
rdfs:Datatype	The class of RDF datatypes.
rdf:Statement	The class of RDF statements.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:Container	The class of RDF containers.
rdfs:ContainerMembershipProperty	The class of container membership properties, <code>rdf:_1</code> , <code>rdf:_2</code> , ..., all of which are sub-properties of 'member'.
rdf:List	The class of RDF Lists.

Table 2.1: RDF classes according to W3C [13]

2.2.2 SPARQL

SPARQL or SPARQL Protocol and RDF Query Language, is the language that has been standardized from W3C, in 2008 [14] [15], for querying RDF data. "SPARQL is essentially a graph-matching query language." [15]. The official recommendation of SPARQL is composed from 3 other recommendations which are [14][16] and [17]. Those three recommendations describe the language itself, the structure of the returning results, which is XML, and the protocol in which the query is submitted to a remote query processor service. "The benefit of having a query language such as SPARQL is also obvious. To name a few, SPARQL is useful:

- to query RDF graphs to get specific information;
- similarly, to query a remote RDF server and get streaming results back;
- to run automated regular queries against an RDF dataset to generate reports;
- to enable application development at a higher level, i.e., application can work" [11]

Property name	comment	domain	range
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:member	A member of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:first	The first item in the subject RDF list.	rdf:List	rdfs:Resource
rdf:rest	The rest of the subject RDF list after the first item.	rdf:List	rdf:List
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

Table 2.2: RDF properties according to W3C [13]

with SPARQL query results, not directly with RDF statements. SPARQL queries, as RDF, are structured in triples. The syntax of SPARQL is composed from 3 blocks [15] which are:

- the WHERE clause. The WHERE clause is composed by a graph pattern which can be used as a filter as well
- the FROM clause. The FROM clause is used to specify the sources or datasets, but in the most recent editions of SPARQL this may be omitted.
- the last clause can be formed in four different ways: [11][15]
 - SELECT
 - ASK

- DESCRIBE
- CONSTRUCT

In our application the only component that is used is SELECT that's why we are going to present the other briefly and extend a bit more on SELECT.

ASK query returns a boolean value depending on whether the given pattern matches the graph pattern of the database. DESCRIBE query is used to add information to the existing data graph. "CONSTRUCT query is another query form provided by SPARQL which, instead of returning a collection of query solutions, returns a new RDF graph." [11]

SELECT is the most common component of SPARQL. It is the component that is used to construct standard queries. The structure of a simple SELECT is the following: The first two components are optional and help to define URI abbreviations. The first component is the BASE directive, which is used to resolve relative URIs. The second component is the PREFIX component. This one used for labeling URIs and using the labels as references to the original URI. Next is the SELECT component. SELECT is the place where the users can declare the attributes, of the entities, that they want to be returned from the query. The attributes are declared with the question mark "?" and the name of the attribute (?attribute). If the users want to have all the attributes, the star character (*) must be used. The next component is FROM. With that clause the user specifies where the data will be retrieved. And finally the WHERE component which is the component that contains the graph patterns that specify the desired results. Inside the WHERE component the user can use the OPTIONAL component, which as its name implies would describe a pattern which is not necessary to be contained in the entities of the query, but if it is contained it will be presented. Besides the OPTIONAL component there are also some other components that are used for filtering. In our application we used two of the filtering options. One is the normal FILTER component where the user can set attributes to be compared with constants, other attributes or arithmetic values. The comparison can have the normal values equal, greater than, less than etc. The second one is a regular expression where the component is almost similar FILTER REGEX followed in parentheses by the name of the attribute and the regular expression in quotes. After that the user may use some other components which are also optional. We will present only the two that we made available for our application. Those two are the ORDER BY component which is ordering the result in ascending or descending sort of way depending on what it was specified by the user. The next and last component that we will refer here is the LIMIT component which limits the results of the query to the number which the user has given. So given a number N if the query has a limit N component then the result will be N lines, which are the first N lines that would have been presented if the limit component was not there.

Here are some examples of SPARQL code that are automatically generated from our application. For the RDF schema the reader can see the application's directories.

```
PREFIX uni: <http://localhost/uni.rdf#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?t0_surname ?t0_forename
  WHERE {
    ?t0 a uni:teacher.
    ?t0 uni:surname ?t0_surname.
    ?t0 uni:forename ?t0_forename.
  }
```

Here the query states that we want to retrieve all the forenames and surnames from the teachers, from the rdf schema uni.rdf

```
PREFIX uni: <http://localhost/uni.rdf#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c0_code
  WHERE { OPTIONAL{
    ?c0 a uni:course.
  }
  ?c0 uni:code ?c0_code.
}
ORDER BY ?code
LIMIT 5
```

In the previous example we want to retrieve the first five courses, showing the code of those that have, and being ordered by the code.

3

Visual Navigator for SPARQL

In this chapter we are going to present the full functionality of the VN and also its limitations and we will try to give a better understanding of the application to the reader.

3.1 Implementation

As mentioned in chapter 2 the application is implemented in Java. Here we dive a bit more in the application and how it is built. The interface of the application has been made with Java swing and whenever a graphic, like a picture, is needed the library awt of Java is used. Whenever the user needs to execute a query the application uses Jena library. Jena is a Java library for RDF , for more information the reader may refer to [18]. The application can be divided in three major parts the Query GUI, which is the the start-up window, the Entity Editor, where the E-R diagram is presented and the formation of the query takes place, and the Result Window, where the queries are executed and the results are shown.

3.1.1 Query GUI

The Query GUI is the welcome screen of the application. Here is implemented the method that parses the RDFS and creates the objects, that later will be used for the formation of the E-R diagram. The way that we implemented the parsing is solely based on XML parsing and we used the special RDFS tags to get the information that we needed. Depending the tag the application decides which instances are going to retrieved, and accordingly creates the entities and adds them to the application's schema. First the program creates the entities. The tag that is used to separate the entities from relationships and properties is the `<rdfs:Class>`. Whenever a `<rdfs:Class>` tag is encountered then the program checks if it has any children. If the tag has children this means that it is a subclass of another entity. In that case the program creates the

entity and connects it with its superClass. If the tag does not have any children the application just creates the entity. The second step is to find all the `<rdf:Property>` tags to create the relationships and the properties of the entities. To identify when an `<rdf:Property>` tag is a property or a relationship we had to look their children. The first tag `<rdfs:domain>` in both cases shows the entity which the relationship or property is related. In the second tag, `<rdfs:range>`, if the tag is defining a property then the last inner tag contains `&rdfs;` label and after that shows the type of property (e.g. Literal). If it defines a relationship then the inner tag contains the label `&schema` and declares the second entity of the relationship. Finally to connect the RDF:properties with the RDF:classes and their subclasses there was a need to create recursive methods, since the RDF schema does not contain all the instances of one class, because it uses superclasses as well.

Another utility of the Query GUI is the execution of the query. To execute SPARQL queries we are using a specialized java library for that, Jena. The reader can find more information at [18]. When the user submits one query the application instantly creates a new .txt file where it stores the query. Using this new formatted file and the RDF file, methods from the library are called and Jena will return the results of the query in a special format. That result will be stored in a new text file for future processing.

3.1.2 Entity Editor

This window (Figure 3.1) is divided in three parts, the E-R diagram window, which is the place that the E-R diagram is presented, the button panel, where control buttons are located and the last is the Query panel, where the query constructed by the user is shown. To create the E-R diagram, as we mentioned already, the application uses its schema, where the entities and the relationships are stored from parsing. In the E-R diagram the entities are represented as ellipses and the relationships are represented as lines connecting the two entities and having a circle in the middle. In this panel by interaction the user can create the queries, and also can change the position of the entities and relationships in the E-R diagram according to his/her needs.

While interacting with the E-R diagram and creating queries the user can also see instantly the results of his/her actions in the Query Panel. The Query panel is also interactive. This makes the query formatting both easier and more expressive. Lines can be deleted, further modified to contain filters or adding properties from entities can be done. To properly add filters the user has to add some modifiers. This can be done from the Expression Editor window (Figure 3.2). In our application we used two of the filtering options. One is the normal FILTER component where the user can set attributes to be compared with constants, other attributes or arithmetic values. The comparison can have the normal values equal, greater than, less than etc. The second one is a regular expression where the component is almost similar: FILTER REGEX followed in parentheses by the name of the attribute and the regular expression in quotes. The "Property Chooser" window appears only in the cases when the user decides to make a filter using the properties. Finally, there are options to ORDER BY and LIMIT the result of the SPARQL query, and CLEAR the canvas in the Button panel.

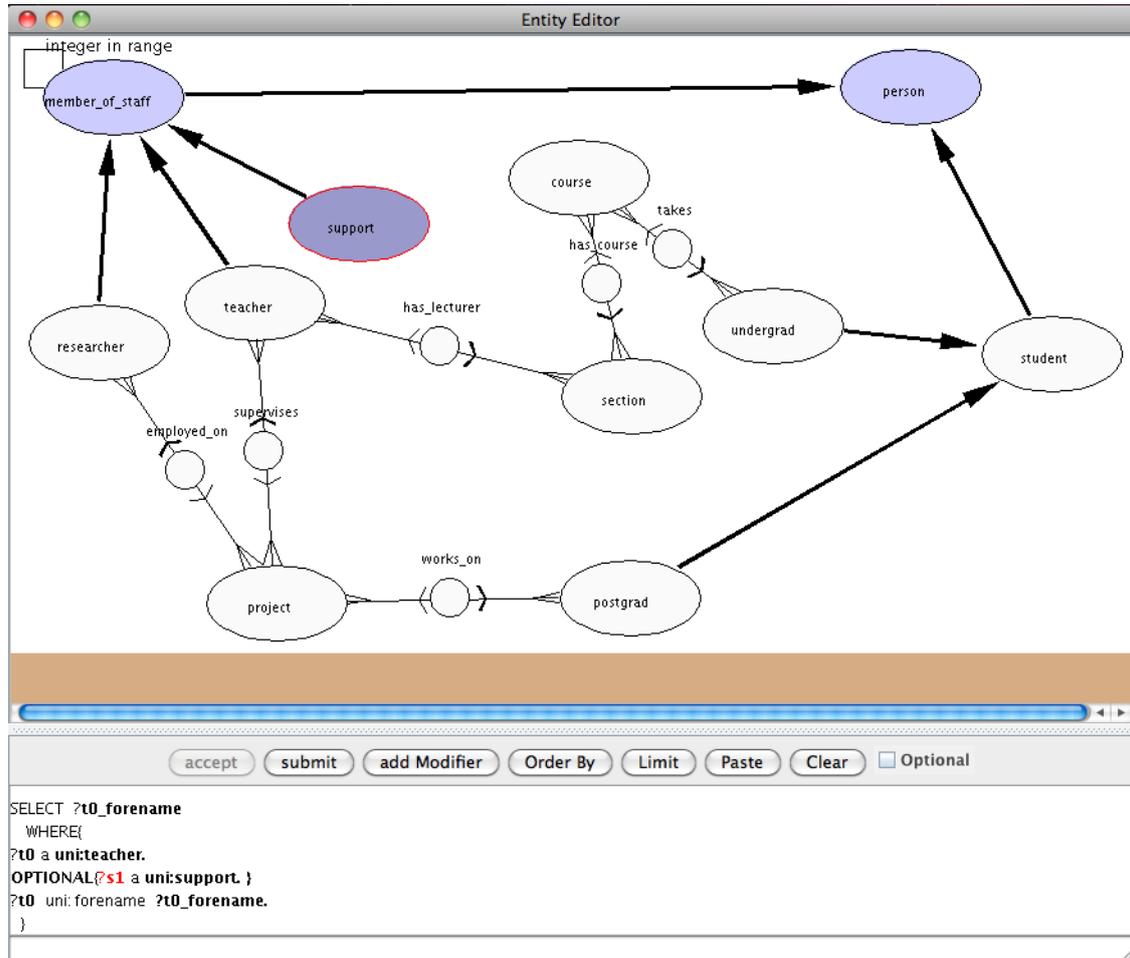


Figure 3.1: Entity Editor window.

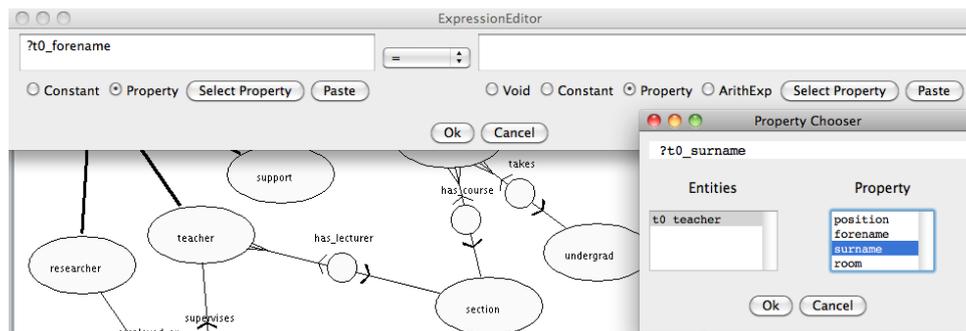


Figure 3.2: Expression Editor and Property Chooser.

3.1.3 Result Window

In the "Database Response" window (Figure 3.3), we see the results of our executed query. The application takes the text file which contains the special formatted results of the executed query and process them to give them a proper format. Because of that, and because executing the query takes time, in the beginning there will be no results appeared but after a few seconds the results, if any, will appear in a grid. One noteworthy feature of the "Database Response" window, that is kept from the previous version of Visual Navigator, is the "copy-and-drop" facility. This feature allows the users to redefine their queries by selecting and copying values from the result window and dropping them in the query window. The application will automatically create filters and will put them in the proper places in the query, so that a more specialized query will be produced. This shows that complex queries can be created step by step from the users.

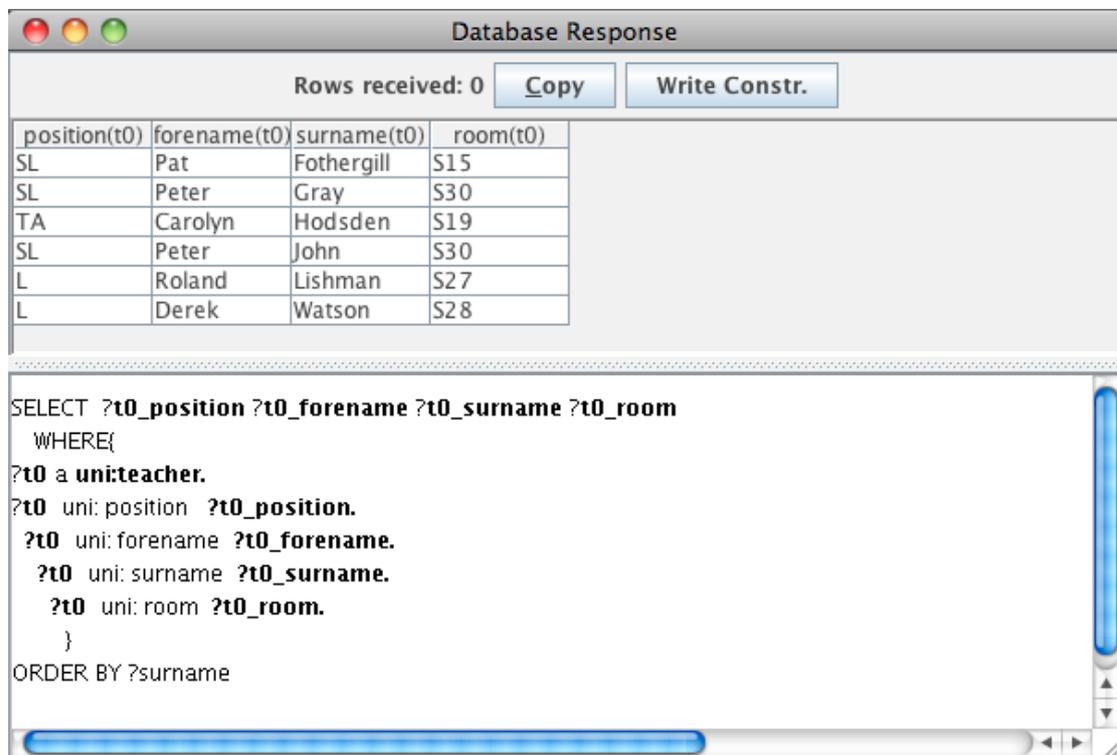


Figure 3.3: Database Response window.

3.1.4 Examples

Here we present some representative queries that show the full functionality and notable features of Visual Navigator.

What we want to retrieve with the first query is the surname, forename and room of the 5 first results of the members of staff, sorted on the surname attribute which their

The screenshot shows a window titled "Database Response" with a status bar indicating "Finished processing". There are two buttons: "Copy" and "Write Constr.". Below the status bar is a table with three columns: "surname(...)", "forename(...)", and "room(m0)". The table contains five rows of data. Below the table is a text area containing the SPARQL query code used to retrieve these results.

surname(...)	forename(...)	room(m0)
Campbell	Colin	S36
Fothergill	Pat	S15
Gray	Neil	S33
Gray	Peter	S30
Hodsden	Carolyn	S19

```

SELECT ?m0_surname ?m0_forename ?m0_room
WHERE{
  { ?m0 a uni:member_of_staff. } FILTER regex( ?m0_room , "S.." ).
  UNION{ ?m0 a uni:researcher. }
  UNION{ ?m0 a uni:support. }
  UNION{ ?m0 a uni:teacher. }
  ?m0 uni:surname ?m0_surname.
  ?m0 uni:forename ?m0_forename.
  ?m0 uni:room ?m0_room.
}
ORDER BY ?m0_surname
LIMIT 5

```

Figure 3.4: Results from first query.

room contains the letter 'S' followed by two character, numbers etc. In Figure 3.4 we can see the result of the executed query and also the code that was used to retrieve those results.

A second example that we are going to see, it refers to a library RDF Schema, and we want to find out the name of the patrons that have loaned a volume with id equal to 1000. In Figure 3.5 we can see the result of the executed query and again the code that was used to retrieve those results.

A third example is in Figure 3.6 and it returns the surnames of all teachers, together with the names of the sections that they teach, and the code of the course of which the section is a part, for fourth year courses only. This example is as well an example in [19] where they used the same schema and we tried to see if our application can construct and execute correctly this query.

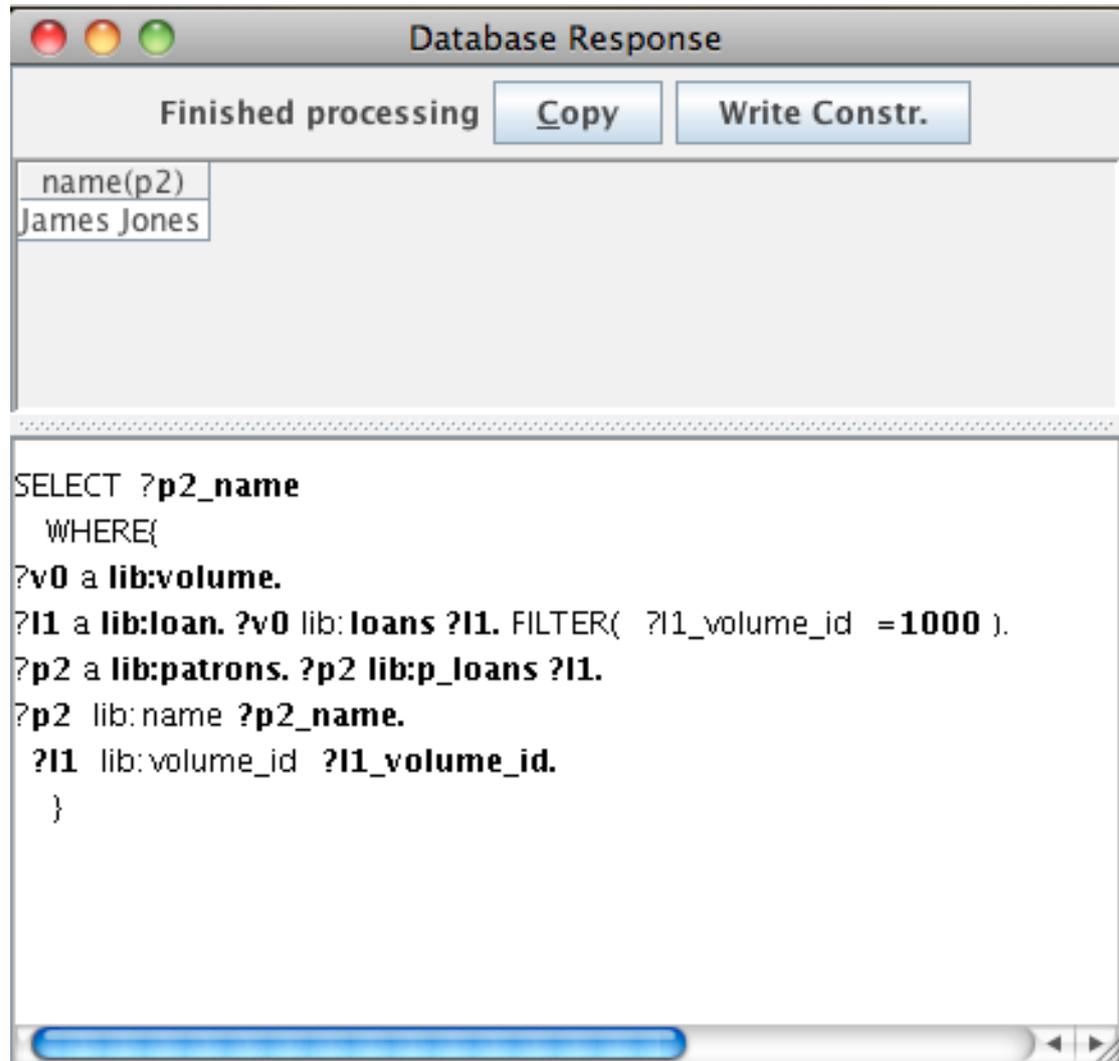


Figure 3.5: Results from second query.

3.2 Limitations

Our application has, of course, its limitations. We didn't implement the full scale of the SPARQL language because that would be time consuming and in some cases impossible to code. One thing that we didn't implement is all the filtering parameters. We only decided to implement the regex filtering option to show that it can be done, in future work someone interested may want to extend it and implement the whole scope of filtering options such as `isBlank`, `isLiteral` etc.

One other major part that we didn't implement is the components `CONSTRUCT`, `ASK` and `DESCRIBE`. We talked about it in the previous chapter. We left those components unimplemented mainly for two reasons, the first reason is because the previous

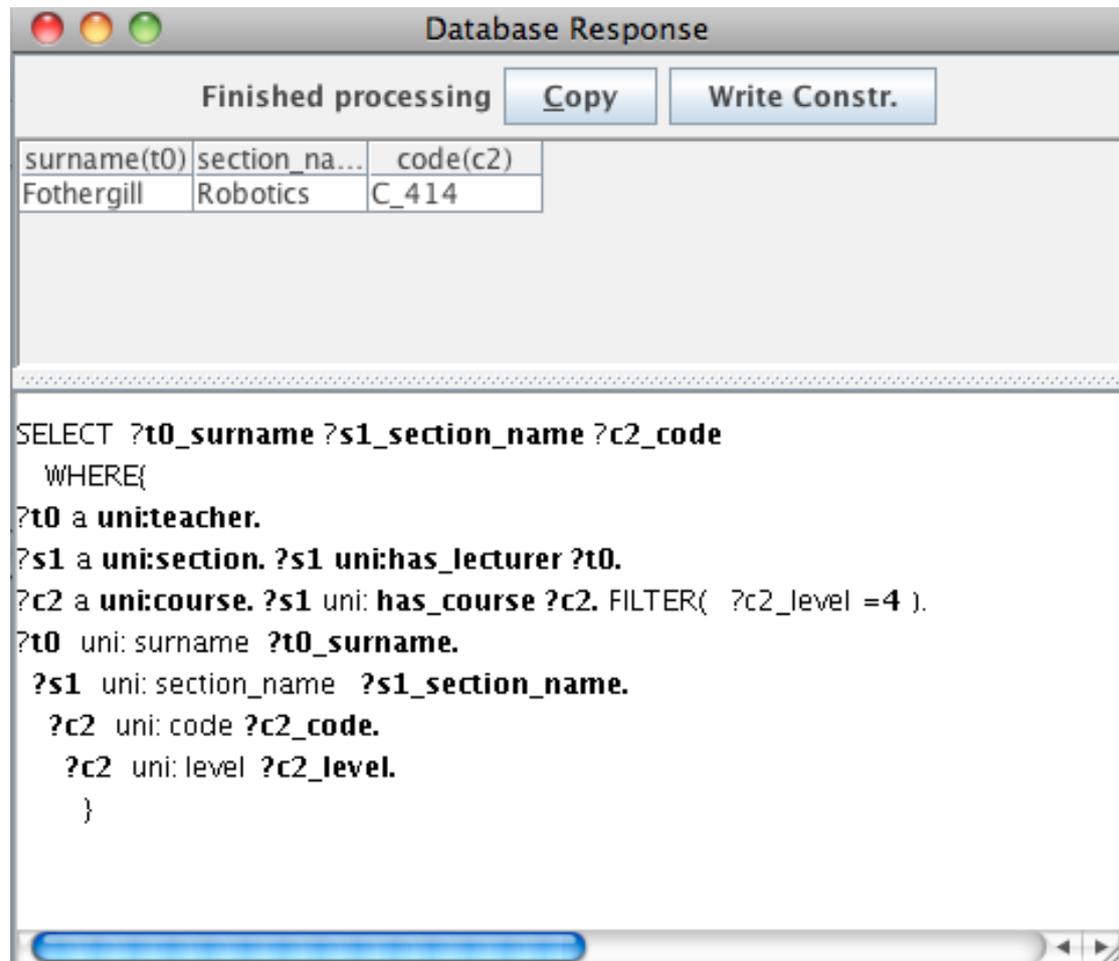


Figure 3.6: Results from third query.

implementation didn't have those features for DAPLEX and based on that the second reason is that we decided our application would be only for constructing queries in SPARQL to retrieve and present data from the database.

A third point that is not properly implemented is the parser. The parser is what reads the XML/RDF schema files and creates the entities with their properties and the relationships. Here the problem is that the features that can be applied and used in such a schema is quite many. To implement such a parser that can work properly for our application it would not serve our purpose. We don't want to make a fully functional SPARQL query constructor, we just want to show that such thing is possible. So we implemented a simple parser that can parse the basic attributes of a XML/RDF schema file and which we believe that it would be able to parse a satisfying number of random files.

One problem that has to do with the coding is that when we are having UNION in the code, the FILTER parameter is not added in the proper place in the Query Editor

canvas. The filter is added properly though in the text file where we save the query, to execute it afterwards. The reason for doing that is the previous structure of the application. We couldn't add the filter to the proper place and manage to have a correct interaction with it.

Another problem, again related with the coding, has to do with adding relationships to the query. We couldn't get the full functionality from the previous application due to inconsistencies with the parsers. You cannot add a relationship if you have not added first an entity, but after that you can add all the existing relationships. The problem is that the relationship should only add the relationships that are related with the added entities.

One assumption that has been made is that all the relationships are many-to-many relationships, because in RDF it is not possible to indicate what kind of relationship is. Those were the major drawbacks that our application has and some were made on purpose while others were infeasible to construct. Still the application works properly for the selected scope.

4

Related Work

As we have already mentioned there is a number of applications in this field, which intend to help users to have a better understanding over the Semantic Web. We are going to classify those works in three categories, in comparison with our own application, and present some of them. The three categories are: applications that create queries in query languages, applications that create conceptual graph queries, and some noteworthy works, which are not similar to our own.

4.1 Browsers creating query language queries

In paper [20] NITELIGHT is presented. NITELIGHT is an application which provides the user with an environment that can both graphical visualize queries and can navigate through ontologies. The language that the queries are constructed is SPARQL, or in more detail is a "visual query language counterpart to SPARQL that we call vSPARQL" [20]. This application is useful for users with partial knowledge of SPARQL and can help them construct queries easier. A user without any experience in SPARQL will not benefit from that application. The major difference between NITELIGHT and Visual Navigator is that NITELIGHT provides an Ontology Browser where the user can browse through to create the queries, while Visual Navigator shows an E-R diagram to the user to help him/her realize the connection between entities and relationships, and create the queries by interacting with them.

Athanasios et al. [21], in 2004, have presented a query generator named GRQL. The purpose of GRQL is to give help to non-expert users in browsing RDF schemas and creating queries in RQL, RDF query language. The user can create queries by browsing through the entities and relationships. One problem though in GRQL, is that we can only see the final query and not the whole procedure of how the query has been constructed. We believe that the educational value of seeing the query while it is constructed is higher, because the user can see the response of his/her actions immediately and also how should

they be put in the correct format to create a query. Finally the construction of the query, in GRQL, is been made by browsing through the entities.

In paper [22] we see a query-by-diagram language called MashQL. The goal is to allow people to build data mashups diagrammatically. The queries are constructed in a language of MashQL but there is an "on-fly" transformation of them to W3C SPARQL. This application allows even unexperienced users to create and execute SPARQL queries, in a variety of sources (That's why it is a mash up tool). Again the main difference with MashQL and Visual Navigator is that there is no E-R diagram in the representation. In paper [23] we see again a tool that helps the user to construct the queries visually. Again we don't see an E-R diagram as our application has, but we see that a type of tree from entities and relationships so the user can select what he/she wants to add in the query. This application is more complete than ours in the SPARQL query department because it includes the whole scope of statements that SPARQL contains.

In paper [24] we see an application "to formulate SPARQL queries for analyzing the semantic data from e.g. the Social Semantic Web. Such data is often large-scale and is collected from a large number of different sources. By supporting a condensed data view and visual query editing as well as browser- like query creation, their VQS allows the users, who are neither familiar with the SPARQL language nor the data structure, to construct queries easily." The main deferences here are that the application is concerned with RDF and not RDFS and that again there is no visual representation of an E-R diagram.

4.2 Browsers for graph queries

QUICK is an application, presented in [25], that helps users construct semantic queries in a given domain. It can take a variety of RDF files but it is used commonly for RDFs. QUICK can create semantic queries by interaction with the user. The user must have some knowledge over RDF and Semantic Web. He/She can give keywords and then with special algorithms and filtering after a number of steps the application will probably result in the desired query.

In paper [26] we see again an application that uses keywords to construct graph queries for conceptual graphs. Here the user starts with some keywords that want to be in the query and with the help of the application, by returning queries written in natural language, the user choses the one that represents the query he/she wanted to ask. The user can refine and modify the keywords in a series of steps to construct a query. And this is the biggest difference with Visual Navigator which creates queries in SPARQL given an E-R diagram. Pradel et. al. [27] extends [26] by creating the queries in natural language as before in the beginning and the queries are transformed to a pivot language which eventually in the end will be translated in SPARQL.

Smeagol [28] on the other hand has an approach which is a bit different from those that we have seen already. With Smeagol, instead of giving general keywords and then trying to specify them you begin with something specific and then try to find more general patterns that match the specific pattern you already have.

4.3 Other kind of applications

An RDF browser, like any normal browser, can work with any content that Semantic Web provides. What [29] presents us is an automated GUI generator which can be used, as mentioned earlier, with a variety of inputs (RDF resources) and can analyze these resources to generate a UI from the available samples. This browser and our application are two completely different things, since Visual Navigator can only have as an input a specific type of RDF schema, cannot create facet, and UIs as the RDF browser does. Also in the browser there is nothing mentioned, even in future work, for creating SPARQL queries, which is the main purpose of VN. We can say for sure that the target group of those two products is quite different, but still the RDF browser is a pretty handy tool, concerning the area of RDF and Semantic Web, so it is something worth mentioning.

Akiyama and Watanobe in [30] present a different application from the others. This application is made for mobiles. The application is an advanced search interface and its purpose is to help the user, even inexperienced ones, obtain information from databases. Even though the application in the beginning does not create SQL queries there are algorithms provided for automated generation of SQL queries.

5

Discussion and Future Work

In conclusion we may say that we achieved our goal. We proved that there is a way to make a graphical application for automatically constructing SPARQL queries. Also we proved, based on [19] that Daplex and SPARQL languages have quite many similarities, so we could use as skeleton the initial Visual Navigator to construct one for SPARQL. Having an E-R diagram really helps the user to visualize the schema and understand better how it is constructed and by making the E-R diagram interactive so that the user can add lines to the query it really makes it easier for any one, even for those without any knowledge on the subject to construct correct queries and retrieve results.

There were points that we need to take a decision on how to continue the implementation of the application. A couple of those points were:

- Where to save the position of the entities and relationships in the E-R diagram. The points in the E-R diagram of each entity and relationship are saved in a .txt file, in the same directory with the application. We also give the possibility to the user to save the new positions of the entities and relationships in the E-R diagram. The reason of having a separate file is because we want to give different users the possibility to have their own representation of data in the E-R diagram.
- Whether to have an undo button. To implement this would require quite big memory to keep track all the actions that user performs. We decided instead of implementing an undo button to implement a clear button, which is a new feature for Visual Navigator. The clear button comes to glue well together with the pre-existed function of deleting a row. So now if the users have built up a big query with plenty of errors they can, instead of removing them one by one, clear everything and start building the query from scratch.

From here on there are some different directions that the project can be continued. First someone can pick-up the project and fix the problems that are existing with the code and were mentioned in the limitations section. Another way that the project can be

expanded is to add the full functionality of the language, adding the ASK, DESCRIBE and CONSTRUCT so that the user can construct all the possible queries and also it would be challenging for someone to try to add all the parameters, or at least the majority of them such as the filtering options, so the application can become more expressive.

And finally, one direction that it would be challenging for someone, is to try and merge both applications from DAPLEX and SPARQL. I believe that is feasible, because while I was coding for VN for SPARQL I had that in mind so I tried to make as few changes as possible. So having two quite similar codes I believe with a bit of effort someone can find a way to merge them and the user when beginning can choose in which languages wants his/her queries so that the proper VN will be loaded.

A

Manual

To start we need to include in our environment the Jena library and execute the file JPFDM.java. "Jena is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and servers" ¹. After we start the application the "Query GUI" opens. We see that there is a text box named "Schema Name", three buttons "Load Schema", "RDF" and "Run SPARQL Query" and finally we see two windows the "Output Window" and the "Constraint Window". The "Schema Name" is where the user will write the RDF schema which he/she desires to load. To load that schema the user presses "Load Schema". After that the user should load the E-R diagram which is done pressing the "RDF" button. The "Constraint Window" is a remnant from the previous application which is not used. The "Output Window" is a window which informs the user of some of the actions that the application does successfully.

After pressing the "RDF" button the "Entity Editor" pops up, showing the E-R diagram of the schema. This window is divided in three parts, the E-R diagram window, which is the place that the E-R diagram is presented, the button panel, where some control buttons are located and the last is the Query panel, where the constructed by the user query is shown. In the E-R diagram the entities are represented as ellipses and the relationships are represented a lines connecting the two entities and having a circle in the middle. The panel is interactive, which means that the user can click in the panel and select the entities or the relationships. When a user clicks for example in the entity, the the entity is selected and its border becomes red. When that happens in the very bottom of the query panel we can see that in the text box is written the line that will be added in the query if the user decides to do that. Adding a line to a query can be done in two ways either by right clicking or by pressing the button "accept". If the user wants the entity to be optional he/she must have checked the "optional" checkbox in the button panel. Now the user has successfully added a line in the query and the result is

¹<http://jena.apache.org/index.html>

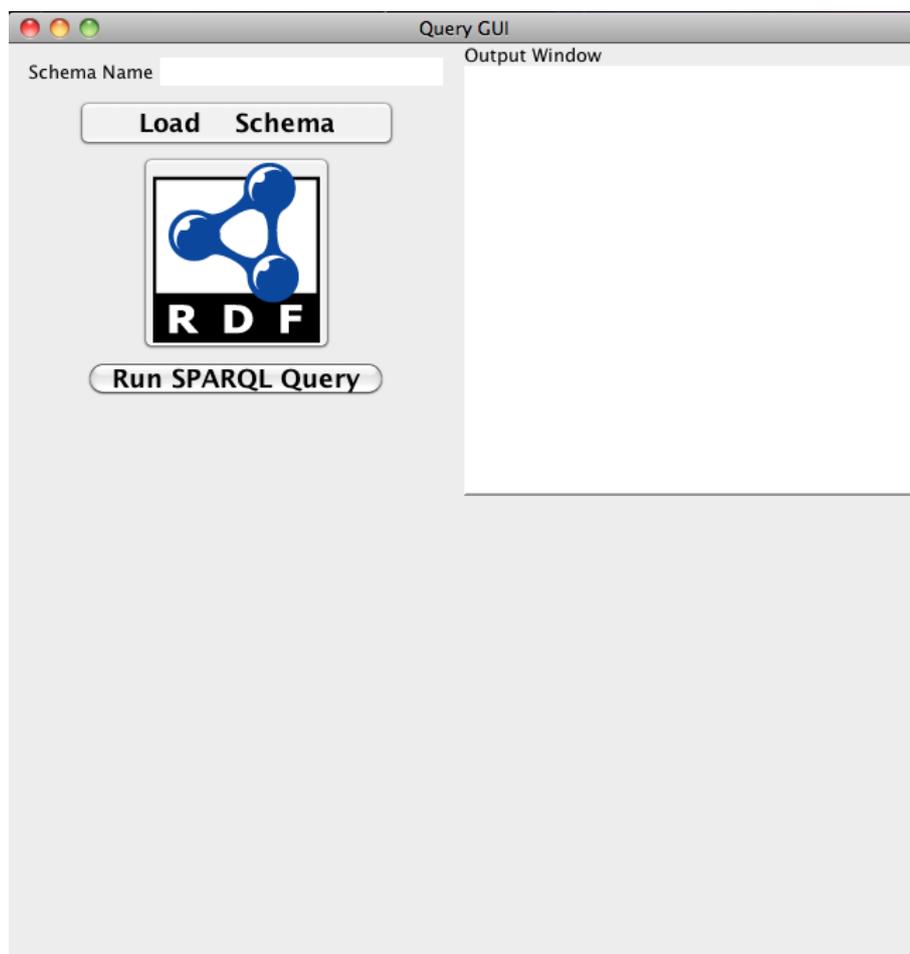


Figure A.1: Query GUI.

shown in the query panel. From now on the user have a variety of options to decide how to form the query.

First of all we have to say that the query panel is also interactive. The user can select the added entities/relationships and they will become red when selected. If double clicked the entity, a window will pop-up again containing the properties of the entity. From those the user can select which he/she wants to add to the SELECT clause of the query. Filtering can be added by pressing the button "add Modifier". This adds to the current line the filter and a box which the user have to double click to form the filter. By double clicking the box the "Expression Editor" window emerges. The "Property Chooser" window is appeared only in the cases when the user decides to make a filter using the properties and clicking the button "select Property".

Besides filtering the user can here specify if he/she wants the results to be ordered or not by clicking the button "Order By" and selecting the proper attributes in the pop-up window. Also he/she can limit the size of the results by clicking the button "Limit" and

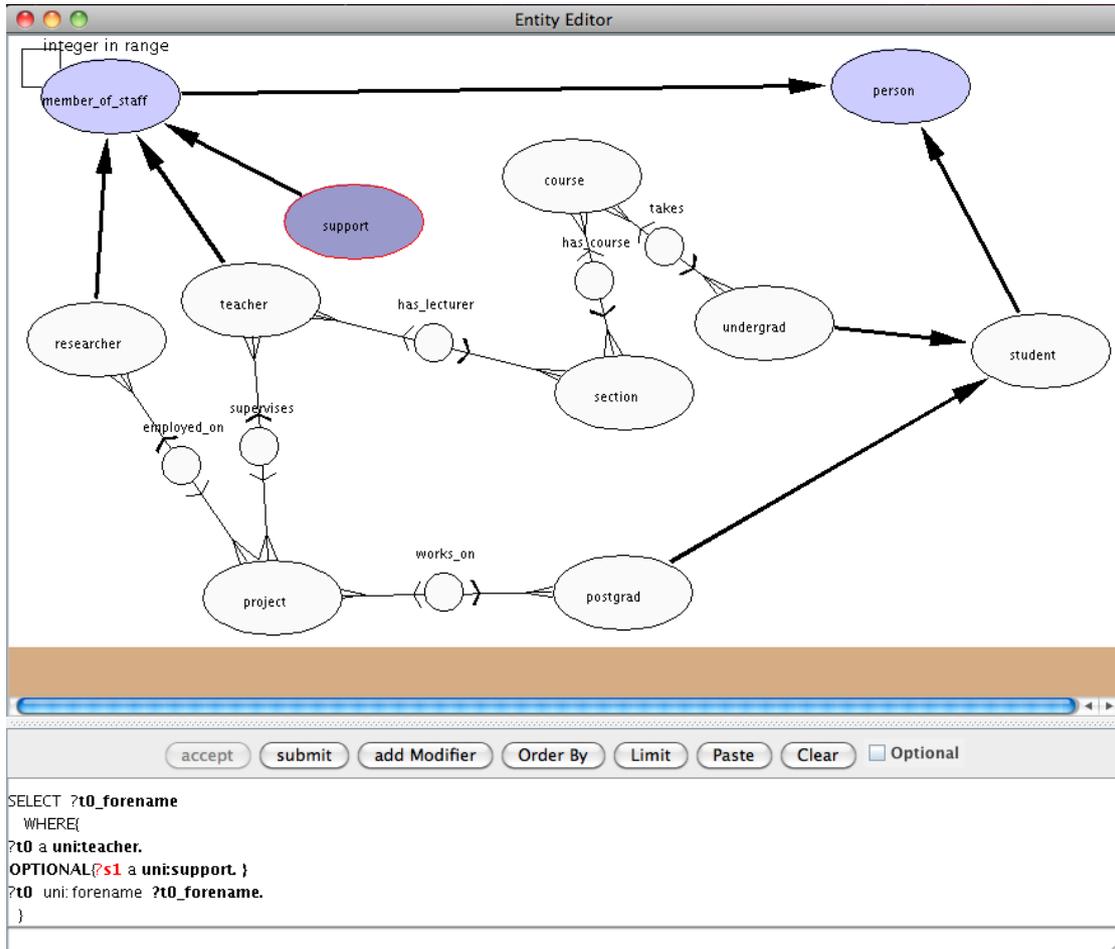


Figure A.2: Entity Editor Window.

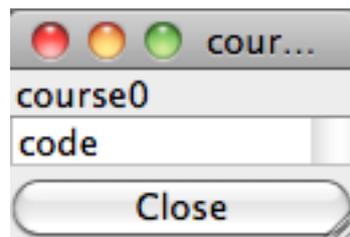


Figure A.3: Attributes window.

putting the desired number of results in the window that will appear afterwards and finally the "Clear" button which clears the canvas and the query. If the user is done modifying the query then he/she can press the "submit" button, which will save the query in a file so in the future the user will be able to run it. By pressing the "submit" button the "Entity Editor" window is minimized and the user will return to the "Query

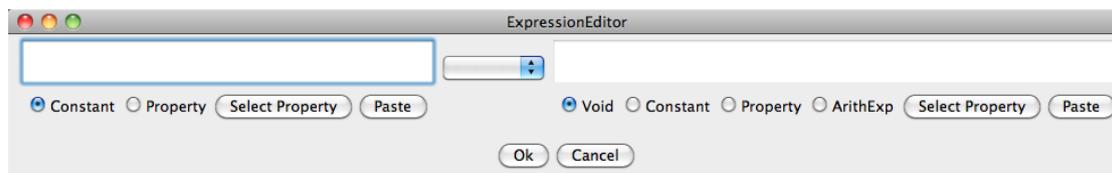


Figure A.4: Expression Editor window.

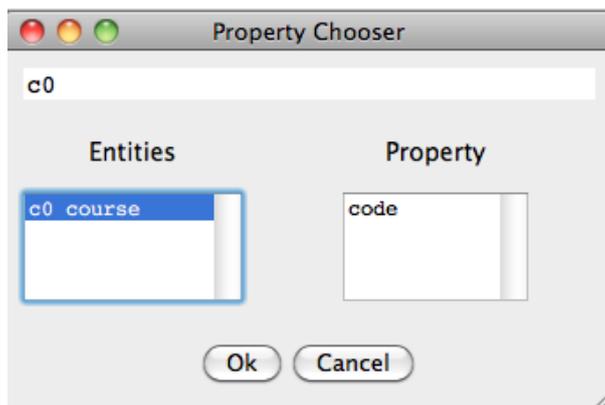


Figure A.5: Property Chooser window.

GUI" window.

Now that we have submitted the query and we are again in the initial window, the user can run the query to get the results. To achieve that the user has to press the "Run SPARQL Query" button, which will process the query and with Jena will run it and get the desired results. To present the results a new window pops-up - the "Database Response" window. Initially there will be no results because the application is still processing the query and the database to get the results but after a few seconds the results, if any, will appear in a grid. The "Database Response" is also consisted from 3 panels. One is the where the current status of the execution of the query and the "Copy" button stand. The "Copy" button is a way of adding filtering in the main query. You can select the desired number of the results to use as a filter, then you press the "Copy" button go back to the "Entity Editor" window and press the "Paste" button. The second panel is the panel that contains the grid with the results, and finally the last panel is a panel which shows the executed query.

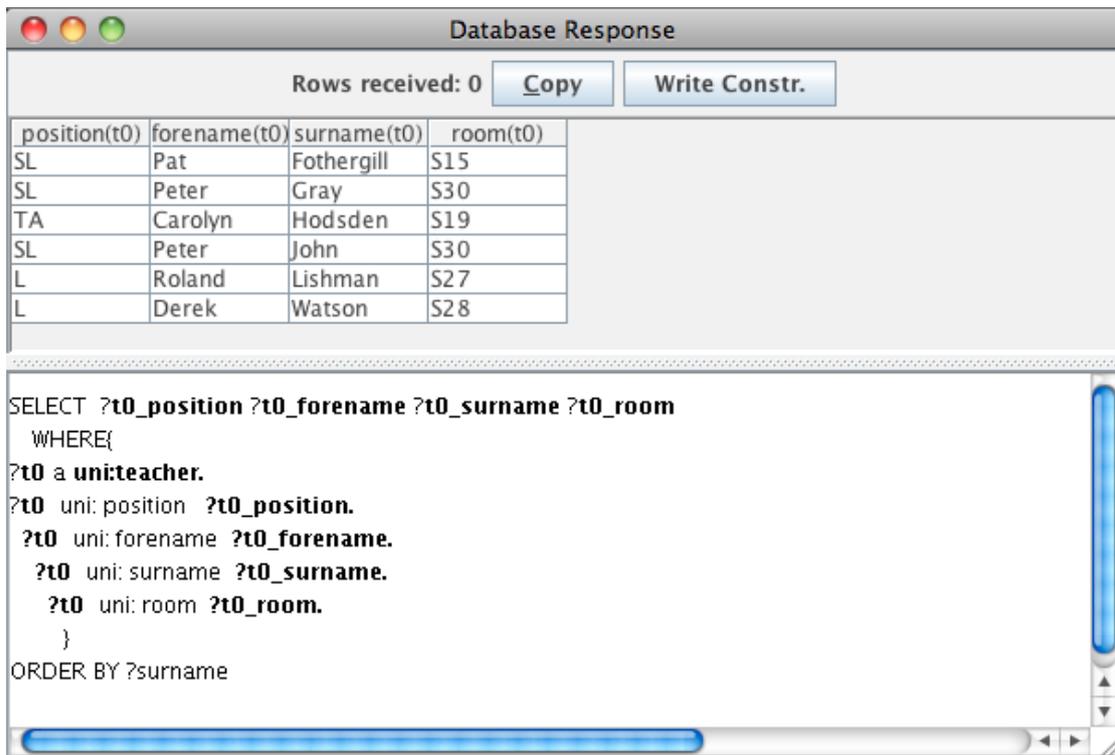


Figure A.6: Database Response window.

B

Implementation

Here we present the essential packages that were made for this application and how the application utilizes them. The list of packages that were made, in alphabetical order, is:

- coolapp
- ereditor
- expressionEditor
- graphSchema
- gview
- parser
- queryEditor
- schema
- scroller
- visresponse

The coolapp package is used for redefining some mouse events and also it used to launch and exit from applications/applets. The ereditor is a really important package. It is used for creating the E-R editor window. In this package files also is the code that creates the 2-D visual representation of the E-R diagram. It gets the entities and the relationships and make their graphical representation, ellipses and circles respectively, in the panel that was already created for that in the E-R editor window. It also contains some features as loading the schema, saving the schema, enlarging the workspace etc.

The `expressionEditor`, as its name implies, is for creating and giving functionality to the expression editor window. It builds the interface with the Java Panel from `java.awt.event`. It also has the methods to construct the property chooser. The property chooser get the entities that are selected for the query and when the user selects one of those for the filter, immediately the gridbox gets filled with its properties. The `graphSchema` library helps to construct some graphical representation such as the ellipses and the circles for the entities and the relationships for the E-R diagram.

The `gview` is again a library for the interface and the interaction of the application, again having to do with canvases and mouse events. The parser library is for parsing the RDF schema. The way that the parsing is done is using the XML structure. From the RDF tags the proper part is retrieved first for the entities and then for the relationships and finally the properties are added to the correct entities. When any of this is retrieved an object is created, depending on what everything is, entity, relationship and property and then the entities and the relationships are added to the application schema.

`QueryEditor` package is the package responsible for the Query Editor window. It makes all its functionality and all of graphical interfaces. There is the parts that create the query and writes in in the window and also in a separate file in applications home directory. There are classes that are responsible for handling filters, deleting entities, filters, attributes etc from the constructed query. Finally there are classes for the creation of the E-R diagram and also for the interaction with all the panels of the Query Editor window. The `schema` package is the package which have the specifications to make the structures for the entities, relationships and also put all of them in a specialized structure to know what the schema contains.

The `scroller` package is for making the scrolling panel and canvas. Finally the `vis-response` package is a general package from the previous implementation. It contains classes responsible some socket models and again some classes that have to do with the interaction design and finally some data structures for the entities that the application uses.

Bibliography

- [1] P. P. Chen, The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems 1 (1976) 9–36.
- [2] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American.
- [3] S. Hawke, I. Herman, E. Prud’hommeaux, W3C Semantic Web Activity, <http://www.w3.org/2001/sw/>.
- [4] N. Shadbolt, W. Hall, T. Berners-Lee, The Semantic Web Revisited, IEEE Intelligent Systems.
- [5] S. Hamby, 2012: The Year of the Semantic Web , http://www.huffingtonpost.com/steve-hamby/semantic-web-technology_b_1228883.html.
- [6] I. Gil, P. M. D. Gray, G. J. L. Kemp, A Visual Interface and Navigator for the P/FDM Object Database, in: N. W. Paton and T. Griffiths (Ed.), Proceedings of User Interfaces to Data Intensive Systems (UIDIS’99), IEEE Computer Society Press, 1999, pp. 54–63.
- [7] E. Sibley, L. Kershberg, Data architecture and data model considerations., Proc. AFZPS Nat. Computer Conf., Dallas, Tex. (1977) 85–96.
- [8] D. W. Shipman, The Functional Data Model and the Data Language DAPLEX, TODS 6 (1) (1981) 140–173.
- [9] S. M. Embury, A Formal Semantics for the Daplex Language, University of Aberdeen Technical Report (AUCS/TR9504).
- [10] P. M. Gray, K. G. Kulkarni, N. W. Paton, Object-oriented databases: a semantic data model approach, Prentice Hall International (UK) Ltd, 1992.
- [11] L. Yu, A Developer’s Guide to the Semantic Web, Springer, 2011.
- [12] G. Klyne, J. J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, <http://www.w3.org/TR/rdf-concepts/>.

- [13] D. Brickley, R. Guha, SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-schema/>.
- [14] E. Prud'hommeaux, A. Seaborne, RDF Vocabulary Description Language 1.0: RDF Schema, <http://www.w3.org/TR/rdf-sparql-query/>.
- [15] R. D. Virgilio, F. Giunchiglia, L. Tanca, Semantic Web Information Management, Springer, 2010.
- [16] D. Beckett, J. Broekstra, SPARQL Query Results XML Format, <http://www.w3.org/TR/rdf-sparql-XMLres/>.
- [17] K. G. Clark, L. Feigenbaum, E. Torres, SPARQL Protocol for RDF, <http://www.w3.org/TR/rdf-sparql-protocol/>.
- [18] B. McBride, D. Boothby, C. Dollin, An Introduction to RDF and the Jena RDF API, http://jena.sourceforge.net/tutorial/RDF_API/#ch-Jena%20RDF%20Packages.
- [19] J. Martins, R. Nunes, M. Karjalainen, G. J. L. Kemp, A Functional Data Model Approach to Querying RDF/RDFS Data, in: BNCOD, 2008, pp. 153–164.
- [20] A. Russell, P. R. Smart, D. Braines, N. R. Shadbolt, NITELIGHT: A Graphical Editor for SPARQL Queries, 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany (2008) 166–176.
- [21] N. Athanasis, V. Christophides, D. Kotzinos, Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL), The Semantic Web ISWC 2004 Lecture Notes in Computer Science 3298/2004 (2004) 486–501.
- [22] M. Jarrar, M. D. Dikaiakos, MashQL: a query-by-diagram topping SPARQL, in: Proceedings of the 2nd international workshop on Ontologies and information systems for the semantic web, ONISW '08, ACM, New York, NY, USA, 2008, pp. 89–96.
URL <http://doi.acm.org/10.1145/1458484.1458499>
- [23] O. Ambrus, K. Möller, S. Handschuh, Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop, in: Workshop on Visual Interfaces to the Social and Semantic Web (VISSW2010), 2010, pp. –.
URL <http://data.semanticweb.org/workshop/VISSW/2010/paper/main/4>
- [24] G. Jinghua, G. Sven, S. Andreas, Visual query system for analyzing social semantic web, in: Proceedings of the 20th international conference companion on World wide web, WWW '11, ACM, New York, NY, USA, 2011, pp. 217–220.
URL <http://doi.acm.org/10.1145/1963192.1963293>

- [25] G. Zenz, X. Zhou, E. Minack, W. Siberski, W. Nejdl, From keywords to semantic queries: Incremental query construction on the semantic web, *Web Semantics: Science, Services and Agents on the World Wide Web*, Volume 7, Issue 3 (2009) 166–176.
- [26] C. Comparot, O. Haemmerlend, N. Hernandez, An easy way of conceptual graph queries from keywords and query patterns, *Conceptual Structures: From Information to Intelligence Lecture Notes in Computer Science* (2010) 166–176.
- [27] C. Pradel, O. Haemmerlend, N. Hernandez, Expressing Conceptual Graph Queries from Patterns: How to Take into Account the Relations, *Conceptual Structures for Discovering Knowledge Lecture Notes in Computer Science* (2011) 229–242.
- [28] A. Clemmer, S. Davies, Smeagol: A Specific-to-General Semantic Web Query Interface Paradigm for Novices, *Database and Expert Systems Applications, Lecture Notes in Computer Science 6860/2011* (2011) 288–302.
- [29] M. Pazienza, N. Scarpato, A. Stellato, Semi-automatic generation of guis for rdf browsing, in: *Information Visualisation (IV)*, 2010 14th International Conference, 2010, pp. 267–272.
- [30] T. Akiyama, Y. Watanobe, An advanced search interface for mobile devices, in: *Proceedings of the 2012 Joint International Conference on Human-Centered Computer Environments, HCCE '12*, ACM, New York, NY, USA, 2012, pp. 230–235. URL <http://doi.acm.org/10.1145/2160749.2160797>