Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2015) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

Using First-Order Logic to Query Heterogeneous Internet Data Sources

Donald Cohen, K. Narayanaswamy*

Cs3 Inc. 5777 Century Blvd, Suite 1185, Los Angeles, CA 90045

Abstract

This paper describes an approach to formulate queries in the language of first order logic over data from disparate sources distributed over a network. The data sources are treated as if they were all in a common database. The data sources may incorporate different stored or computed methods of providing data– web services and REST APIs, XML/JSON repositories, web pages, full featured databases, flat files, etc. We describe the technical foundations and benefits of this approach and compare it to other extant solutions to this problem. This approach enables end users to formulate ad-hoc queries in logic to correlate the data sources. For programmers creating applications, using a logic-based language to specify correlation criteria shortens the development life cycle and reduces cost of maintenance, compared to manually coding the equivalent correlation algorithms in procedural languages.

Keywords: virtual database, first order logic,Internet data sources, ad-hoc queries, data correlation, virtual relations.

1. Overview of Problem and Solution

More and more data is now available on the Internet. Short of manual coding of algorithms, however, there is no practical mechanism to combine and correlate data from different sources. Our approach to this very important problem is to view all of the data available in a relational framework (familiar from mathematical logic and

* Corresponding author. Tel.: 0-310-337-3013; fax: 0-310-337-3012.
E-mail address: swamy@cs3-inc.com

relational databases -- RDBs), and use first order logic (FOL) as the basis for a query language. FOL provides syntax for logical assertions. We extend FOL to support queries similar to those seen in relational databases by using the notation familiar from set theory, where sets are typically described in the form {x s.t. P(x)}, read as the set of x such that P (a predicate) is true of x. Similarly, sets of tuples can be written in the form {x, y s.t. P(x, y)}. See Section 6 for a description of other approaches to this problem and how they compare to our solution.

As a motivating example, suppose you want to find houses for sale within a given zip code that are within 0.25 mile of a day care facility. You might use the Zillow.com web site to find candidate houses. Then, for each house you might copy and paste the address into the YP.com (Yellow Pages) website and look for day care centers close to that address, sorting the results to visually filter out those centers that are more than 0.25 miles away from the house address. This paper describes *TriggerWare*, a system that automates this kind of computational correlation activity. This facility could be used by users for ad-hoc queries or by programmers building complex applications.

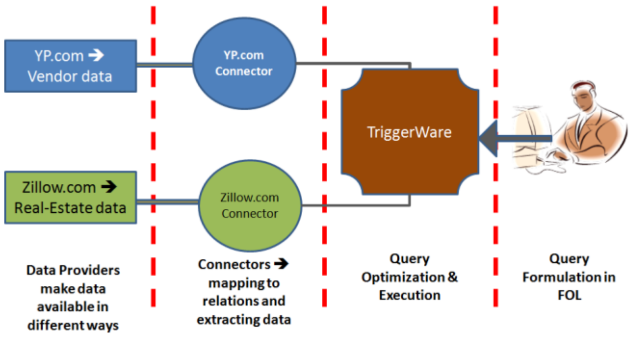
YP.com → Vendor data
YP.com Connector
TriggerWare
Zillow.com → Real-Estate data
Zillow.com Connector
Data Providers make data available in different ways
Connectors → mapping to relations and extracting data
Query Optimization & Execution
Query Formulation in FOL

Figure 1: How the TriggerWare Approach Works

The TriggerWare approach involves encapsulating both the Zillow.com data and the YP.com data as one or more relations, and writing an FOL query involving those relations. In general, these tasks (shown in Figure 1) are not equally difficult, and they are not performed by the same individual:

- **Data Providers:** supply the raw data that people want to use. This data is already abundantly available on the Web via data sources such as web services, APIs, XML repositories, SQL databases, HTML pages, etc. The data from most sources must generally be used "as is" by applications.
- **Connectors:** encapsulate data sources as relations. Connectors must be defined by people in our approach – in contrast to approaches seeking to automate the understanding of the semantics of data sources [1]. Connector creation is a non-trivial task – requiring some data modelling and programming skills. We have many standard connectors available to link to common types of data sources (e.g., relational databases) – but even in these cases our experience indicates that human beings are needed to refine the underlying schema to define useful connectors. A single connector to a data source can be reused and leveraged by many people, each defining many queries. For a typical data source, defining a connector takes on the order of hours. It is important to note that connectors can be provided by a party other than the data provider. Further, more than one party can provide connectors (extracting different relations) from the same data source. Different users can choose the connectors that best match their requirements.
- **Query Formulation:** both users wanting to answer specific questions and programmers building complex applications can define queries in FOL. Queries can be defined in a matter of minutes once users understand the data available through relations, and the questions they want answered.
- **Query Optimization and Execution:** These are the services provided by the TriggerWare framework to actually compute the answer to a query. These services will typically execute automatically when invoked on a constructed query; and the typical query is answered within a matter of seconds.

In addition to the interactive query scenario presented in Figure 1, we expect programmers to embed queries into code, just as they do now with data stored in relational databases. Our approach enormously expands the universe of data sources that can be combined through the use of logic queries. The entire TriggerWare service is itself encapsulated as a set of REST API functions to facilitate programmatic invocation. In this paper we document the essential technical aspects of our approach and show several samples of using logic to query various data sources.

2. TriggerWare Technology Overview

TriggerWare is a virtual database platform – for more details on the core technology see [2]. Data is represented as relations. However, the notion of *relation* in TriggerWare is far more general than in traditional relational databases. A relation is a set of tuples of objects. Every relation has a fixed "arity". All tuples ever in the relation have that arity as their length. Furthermore, every relation must support the ability to test whether a given tuple of that length is in the relation. Many other capabilities are optional and some of those are described below. The "defrelation" facility shown in the examples allows a coherent relation to be assembled from a collection of generic capabilities. TriggerWare is implemented as an extension to the Lisp programming language, and for that reason Lisp syntax will be shown in the examples of the query language, e.g. R(x,y) is written (R x y).

Stored Relations: Stored relations, analogous to tables in relational databases, are those that support direct capabilities to add and remove the tuples in the relation. Stored relations do not play a big role in the subject of this paper, but we mention in passing that TriggerWare allows complete control over the representation of stored data. The examples of stored relations in Figure 2 use the default representation for tuples (a simple list of tuples):

```
;; Person (unary relations -- :arity 1 – are referred to as  "types")
                    (defrelation person  :arity 1)
; Binary  relations between persons – with stated "type" constraints and names for each column
                    (defrelation  father :artiy 2 :types (person#child person#father))
                    (defrelation  mother :artiy 2 :types (person#child person#mother))
```

Figure 2: Examples of Stored Relations

FOL-like Query Language and Defined Relations: The result of a query is actually a set of tuples, i.e., a relation. TriggerWare allows relations to be defined by queries analogous to set theory notation, *(<variable list> s.t. <logical formula>)* where the logical formula is written in the language of FOL, including relations (usually identified by name) applied to arguments that may be either constants or variables, Figure 3 shows examples of defined relations.

```
;; parent relation defined in terms of father and mother relation
          (defrelation parent
             :definition
              ((person parent) s.t. (or (mother person parent)
                                        (father person parent))))
;; grandparent relation defined in terms of parent
          (defrelation grandparent
             :definition
              ((person grandP) s.t.
               (Exists (parent) (and  (parent person parent)
                                      (parent parent grandP)))))
```

Figure 3: Examples of Defined Relations

Since TriggerWare is a Lisp extension, the interface for generating tuples of a relation is integrated into the iteration facilities of Lisp. The LISTOF construct is shorthand for a loop that collects a list of tuples, each tuple represented as a list of objects. In order to retrieve the pairs of people and their parents, you can use the listof construct on the relation parent as below:

```
(listof (x y) s.t. (parent x y))
```

The listof construct also accepts the full syntax of logical formulae. So, one can retrieve the siblings of a person by using the following listof construct ("sibling" relation holding between persons who share a parent):

```
 (listof (x y) s.t. (Exists (P) (and (parent x P )
                                     (parent  y P)
                                     (not (equal  x y)))))
```

The listof construct is analogous to the SQL SELECT statement What is important is that, as with SQL, the same query syntax applies to all relations. This point becomes much more significant as we discuss computed and derived relations next.

Computed Relations: this term is used for relations that never change and are typically computed by code. An example is the relation "<", which is true of two objects when both are numbers and the first is less than the second. Notice that it is possible to test a tuple for membership in the relation <, but it is not possible to generate the tuples in the relation. In order to show other relevant capabilities of computed relations, we use the "+" relation below:

```
;; viewing + as a relation in a query – uses a Lisp function "+" to generate the relation
(defrelation +
        :documentation
                "(+ x y z) is true if and only if x, y and z are numbers and x+y=z."
        :computation (lisp-function + 2 1)
        :types (number number number)
        ;; specifies estimated sizes for outputs given inputs – nil means size is infinite
        :size (    (output output output) nil
                    (output input input) 1
                    (input output input) 1)
        ;; given one of the inputs of + and the result of +, we can generate the other input
        :generator
                ((simplegenerator (a output c)  (and (numberp a) (numberp c) (- c a)))
                (simplegenerator (output a c)   (and (numberp a) (numberp c) (- c a)))))
```

Figure 4: Example of a Computed Relation

The declaration "(lisp-function + 2 1)" specifies that for any pair of objects, "a" and" b", there is at most one "c" s.t. (+ a b c) and that "c" is the result of evaluating the lisp function "+" as follows: (+ a b). If this results in an error then there are no such "c" values. The "2" and "1" in the specification are the number of inputs and outputs to pass to and receive from the "+" function. The construct also creates a tester using the same function, with a size estimate that there is 1 set of outputs for each set of inputs. The additional size and generator arguments indicate that in this case (though not in general for lisp functions), there are also ways (subtraction) to find the single "a" value for a given "b", "c" pair and the single "b" value for a given "a", "c" pair. . One cannot generate all the tuples for the "+" relation described (since that set is infinite). Once computations are represented as relations, these relations can appear directly wherever relations are allowed – in queries, in logical formulae, etc. The computational functions in Lisp can all be represented as relations as above, and used in queries.

Derived Relations: are similar to computed relations except that they are not required to be static. Specifically, they are allowed to depend on other relations, in the sense that they are allowed to change only when those other relations change. Relations defined by FOL are technically derived relations (derived from the relations that appear in the definition), but they are such an important case that they are given their own syntactic category in TriggerWare. The definition in FOL can be used to compute all sorts of capabilities that, in the more general case have to be supplied explicitly. Examples of derived relations include powerful idioms such as transitive closure of a relation, min, max, sum, etc.. Note that transitive closure cannot be expressed in FOL. However, since the tuples of the transitive closure relation can be computed in a Lisp function, we can define the transitive closure relation using a derived relation with that computation as the relation representation. And, once the transitive closure is so defined, we can use it in queries as we would any other relation. Derived relations provide the foundation to query Internet data sources, and we elaborate on them next.

3. Using Derived Relations to Interface with Arbitrary Data Sources

Data available these days within an enterprise or on the Internet fits into the category of derived data in an interesting way. Normally, we use derived relations to describe how the tuples of a derived relation depend upon other relations. However, the relations like those that represent Zillow data or YP.com data are completely out of our control. We cannot tell when these data change or whether the queries to them are running in a coherent state that is fixed for the duration of query execution. This is already inevitable when people or programs try to combine data from different data sources manually or through code, and will also be true for our solution.

We return to the motivating problem of finding houses in Zillow that are within 0.25 miles of day care centers. To consider using TriggerWare to answer this question, we would need TriggerWare to be connected to the Zillow data source. That would let us use the Zillow data through theTriggerWare query language. It is important to remember that the Zillow data source does not provide a list of ALL homes. Once has to provide a location in order to get the homes near that location. So, the zillow-homes TriggerWare relation must be declared to reflect the fact that all one can really generate through the Zillow API as output is a list of homes for a given input location. The tuples of the zillow-homes relation are generated by calling the function "**generateZillowTuples**" (highlighted in bold here and below for convenience).

```
 (defrelation  zillow-homes
   :derivation individual-derived  ;; means that this relation has only the capabilities listed in this form
   :documentation "(zillow-homes location address neighborhood type price bed bath sqft ybuilt lotsize) means that
the Zillow.com API, when provided "location", returns entries relevant to that location including the given address,
neighborhood, type,  <some documentation elided...>  "
   :arity 10   ;; the number of "columns" in the relation – also deducible from the number of :types below
   ;; ":types" provides both the types and mnemonic column names for users to understand the relation
   :types (string#location  string#address  string#neighborhood string#type number#price  number#bed
                             number#bath    number#sqft   number#ybuilt    number#lotsize)
  ;; the size estimate that this connector provides only the first 100 results from Zillow
  :size ((input output output output output output output output output output)  99)
   ;;  generator spec that given the "location", generateZillowTuples will generate a list of
   ;; of homes with address, neighbourhood, etc. 1e6 is the very approximate "cost" estimate
  :generator      ;; explained further below
   ((simplemultiplegenerator
     (location output output output output output output output output output)
     (and (stringp location) (ignore-errors (generateZillowTuples location)))
     1e6))))

(defun generateZillowTuples (location &aux  url  file)
    ;; code to contact the Zillow API with the location and return the list of homes as below:
    ;; list containing (address neighborhood type price beds baths sqft  ybuilt  lotsize)
)
```

Figure 5: Defining the Zillow-Homes Relation in TriggerWare

Note that the only access provided by the Zillow site is the one in which the first argument, the "location", is provided as input, and all the other columns are generated from that input. The real work in implementing this connector is in the **generateZillowTuples** function used to generate the results for a given location argument. This is done by invoking the appropriate function from Zillow.com API with the location, parsing the data, and retrieving the data relevant for the Zillow-Homes relation. We believe that these programming details are simple enough that we do not need to describe them here. The *SimpleMultipleGenerator* used in Zillow-Homes is similar to the *SimpleGenerator* specification used in the "+" declaration except that it allows multiple results to be returned, and accepts a cost estimate for the operation (in this case estimated as a 1e6, meaning one million, in units not worth going into here). Incidentally, we think it is important to mention that the testing code for Zillow-Homes just uses this generator to find all of the results and searches them to test for matches of the input columns provided.

In exactly the same way as above, we can define YellowPages as a relation that encapsulates the YP.com API data as follows: *(yellowpages category location name address city state zip miles lat lon)*. This means that the YP.com, API when provided "category" and "location" returns entries with the names, addresses, and other attributes corresponding to the vendors for that category and location.

4. Logical Queries over Internet Data Source with Sample Results

Figure 6 shows the query that answers the specific question in our motivating example. The main query is a list of conjuncts over two virtual relations. We have already described these two major data source relations – ZILLOW-

HOMES and YELLOWPAGES. The query uses these relations to find the homes for a given zip code (90064 in this case). We then use the address returned from Zillow-Homes (HOMEADDRESS) as the location for the YellowPages relation to search for day care centers. Next, we filter the results for distance less than 0.25 miles.

5. Query Execution and Optimization

In general, there are many algorithms that could be used to answer queries. The TriggerWare query optimizer is analogous to those used by RDBs, but of course it works on a different query language and a more general model of relations. Details of how this works appear in [3]. The summary is that the query optimizer searches a space of potential algorithms based on the capabilities provided by the underlying relations and additional declarations about sizes of generated tuple sets or cost of invoking the generators. At each stage the optimizer estimates the cost and size (number of output tuples) based on cost estimates provided with the individual generating capabilities and size data provided with the relations (or default assumptions if these are not provided). The algorithm executed is simply the one with the lowest estimated cost.

```
((HOMEADDRESS  BED  BATH  SQFT   DAYCARENAME   DAYCAREADDRESS   MILES) S.T.
 (Exists (NEIGHBORHOOD1  PRICE  TYPE1  YBUILT1  LOTSIZE1  CITY1  STATE1  ZIP1  LAT1  LON1)
      (AND
          (ZILLOW-HOMES "90064"  HOMEADDRESS  NEIGHBORHOOD1  TYPE1  PRICE  BED  BATH
                                                              SQFT  YBUILT1  LOTSIZE1)
         (YELLOWPAGES "daycare" HOMEADDRESS  DAYCARENAME  DAYCAREADDRESS  CITY1
                                                       STATE1  ZIP1  MILES  LAT1  LON1)
         (<= MILES 0.25))))
```

Figure 6: Logic Query Combining Data from YellowPages and Zillow

The query of Figure 6 produces the results in Table 1 – we show only the first 6 results (there were more).

HOMEADDRESS	BED	BATH	SQFT	DAYCARENAME	DAYCAREADDRESS	MILES
2491 Purdue Ave APT 203, Los Angeles, CA	2	1	986	Wonder Years Preschool	2457 Sawtelle Blvd	0.17725079
2491 Purdue Ave APT 203, Los Angeles, CA	2	1	986	Circle-Friends The Path To	11508 W Pico Blvd	0.20338617
2491 Purdue Ave APT 203, Los Angeles, CA	2	1	986	Richland Avenue Preschool Center	2623 Coolidge Ave	0.23041677
2473 Amherst Ave, Los Angeles, CA	2	2	913	Creative Learning Preschool	2523 Armacost Ave	0.15308192
2473 Amherst Ave, Los Angeles, CA	2	2	913	Inglewood Community Adult	11910 W Pico Blvd	0.22653708
2204 S Bentley Ave APT 304, Los Angeles, CA	2	3	1293	Samuel Goldwyn Foundation	2114 Pontius Ave	0.170606

Table 1: Results of the Execution of the Sample Query

Size and performance specifications guide the TriggerWare compiler in selecting the optimal algorithm for a query. Different algorithms may return different results, but these results represent the same relation (i.e., the same set of tuples). Thus, for example, one result might be in a different order than another. The meaning of the query does not depend on such things as relation representation or size or cost estimates. The result is guaranteed to match the meaning of the query. Therefore, if one has control of the relation definition, one can develop a working query independent of performance considerations, and then optimize the query **without affecting its correctness** by altering the representations, size estimates, and cost estimates of the underlying relations. We consider this a critical advantage of the TriggerWare framework for logic queries. While RDBs provide similar capabilities for *stored* data, they provide no way to handle the variety of data sources we discuss in this paper. Navigational languages like XQUERY [4] and XPATH [5] do not provide query optimization. They force the programmer to encode the "optimal" algorithm in the details of the query itself, similar to what one does in writing procedural code.

In the TriggerWare approach it is possible that no algorithm exists for certain queries. This is an inevitable consequence of supporting relations that do not provide all the possible relational capabilities, as often happens when encapsulating Internet data sources. So, if a query requests all of the Zillow-Homes data for all possible locations or even all of the possible "locations" where homes are available, that query cannot be answered. In such cases, TriggerWare will print a message that this query cannot be generated while trying to compile the query.

6. Comparison to Other Approaches and Relative Benefits/Drawbacks of TriggerWare

There are several related technologies that provide solutions to different variations of the problem we describe in this paper. We wish to delineate these and compare them to our approach in more detail.

6.1. Relational Databases, SQL, and Data Warehouses

Our work has much in common with RDB technology, including the underlying concept of relations for representing data, the use of a non-procedural query language and an optimizer. The critical difference is that TriggerWare supports a far more general notion of "relation", enabling the framework to encapsulate a wide variety of data resources as relations, including those found widely on the Internet. Specifically, we have made use of the ability to provide arbitrary code for generating "slices" of the data, and the fact that the ability to generate ALL of the data in a relation at once is NOT required.

Another difference is our choice of query language. We prefer FOL to SQL because it is much simpler, more purely relational, and easier for users. Our view is that the parts of SQL that are missing, (what some might consider to be lack of power of FOL), are those that are not relational at all and are related to issues other than defining the set of results that meet the query criteria. In particular, here are some major differences that lead us to prefer FOL:

- In FOL there are no null values because they are not needed; SQL, of course, uses null values, greatly increasing the complexity of the language, such as including null tests, different types of joins, etc.
- Functions are subsumed by relations. Although we can support a more functional syntax, expressions like x+y>0 are translated into something like (Exists (z)(and (+ x y z)(> z 0)))
- Relations, by definition, cannot contain duplicate rows (whereas table and the results of SQL queries can). Therefore, in FOL, there is no need for things like "*select distinct*".
- Some SQL constructs like "*group by*" can be handled by relational means (recall sum, max examples described earlier);
- Non relational SQL constructs like *limit* and *order by* are better handled by means outside the language;

Our work is also related somewhat to data warehousing. Typical data warehouses use the ETL (Extract-Transform-Load) paradigm to redundantly store data extracted from a set of other databases and data sources. The redundant database is organized and optimized for offline analytics and reporting. Data warehousing simply will not work in cases where all the data is not available from the data sources (this is true of most APIs/Web Services). APIs may not provide all the data available. The data copied to the warehouse will always be out of date with respect to the "real" data. Warehouses provide large-scale, high-performance offline analytics and reporting. TriggerWare offers a relatively lightweight, cost-effective alternative to answer correlation questions about real-time data in real-time while largely leaving the data in place within the original data sources.

6.2. Semantic Web Initiatives

The Semantic Web activity of the W3C [6] is related to our work. Quoting from [6], "…*the idea of having data on the web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications…*". The vision is compelling, and one we support enthusiastically. The OWL Web Ontology Language [7] and Resource Description Framework (RDF) [8] aspects of the Semantic Web provide XML tags to specify ontologies and resource descriptions. The latest Semantic Web initiative is Linked Data [9]. The purpose of this initiative is to link up the structured data from many different places. SPARQL [10] is the language used to query data sources described in this kind of framework that is based on OWL/RDF. SPARQL assumes ALL the data is generable from the data source – this is not true for data sources like web services and APIs. SPARQL is much more complex than FOL, and shares many of the drawbacks of SQL that we have described in Section 6.1.

There are many researchers working on automating the acquisition or learning of semantic models of web content [1]. In contrast, we are *not* trying to automate reasoning about the *semantics* of data sources (i.e., automatic creation of data source connectors). While TriggerWare includes automatic connectors to commercial databases and some simple APIs, in general, we rely on people to understand the data source semantics and encapsulate the data provided as a set of relations that will be easy for other humans to understand and use. TriggerWare supports composition, correlation, and integration of data sources once the connectors are defined by humans. We will definitely leverage advances in automated acquisition of data source semantics as relevant in the future.

6.3. Using Query Languages with Web Services

Web Services Description Language (WSDL) is an XML-based mechanism to specify the functions provided by SOAP and REST web services. One can use query languages like XQUERY [4] to combine information from different web services. XQUERY is based on the same model as XPATH [5], and exploits the tree model of the information content of the data. This leads to a navigational style of specifying queries where the user has to specify algorithmic choices that need not be (in fact cannot be) expressed in the FOL-based query language that we use in our framework. In our approach, the user focuses solely on WHAT is being retrieved without regard to how it is computed. The TriggerWare framework provides a query optimizer that will select the best algorithm automatically given the properties of the data sources involved. Navigational languages result in queries that are dependent upon the specific structure of the artefacts returned as results by the web services. We consider this a big drawback.

7. Conclusion

TriggerWare is a platform that provides *pragmatic and practical* answers to questions about one or more Internet data sources that are currently, at best, very tedious to answer. TriggerWare already incorporates built-in connectors for many popular web sites, web services and APIs, and most commercial database products. We are in the process of commercializing this product in different markets. TriggerWare is packaged as a W3C compliant web service to make it convenient for programmers to use logic-based queries. For typical end users, we see the need for appropriate graphical or natural language facilities to support the creation of ad-hoc queries in FOL. Our focus has been largely on the TriggerWare engine and infrastructure necessary to support logic queries over data sources.

For our public facing websites, we limit the amount of execution time to make sure that users do not inadvertently define computationally intensive queries. In theory, we could let all the queries run to completion to generate ALL the data provided by the underlying data source APIs, but that may be prohibitively expensive. Many data source APIs also restrict the amount of data provided as part of their “Terms of Use”. Therefore, the data computed as the answer to a query may be only a subset of the ideal or complete answer. When presenting query results to the user, these computational and API restrictions have to be clarified as additional context. This context should help users in interpreting the results before they make decisions based on the results they have been presented.

References

[1] Taheriyan, M., Knoblock, C.A., Szekely, P., Ambite, J.L., A Scalable Approach to Learn Semantic Models of Structured Sources, *Proceedings of the 8th IEEE International Conference on Semantic Computing (ICSC 2014*).
[2] Cohen, D.N., The Ap5.manual, http://www.ap5.com.
[3] Cohen, D.N., Automatic Compilation of Logical Specifications into Efficient Programs, *American Association of Artificial Intelligence (AAAI) Conference*, 1986.
[4] XQUERY 3.0 Specification, http://www.w3.org/TR/xquery-30/
[5] XPATH Introduction, http://www.w3schools.com/xpath/xpath_intro.asp
[6] Semantic Web, http://www.w3.org/standards/semanticweb/ & http://www.w3.org/2001/sw/
[7] OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/
[8] Resource Description Framework (RDF), http://www.w3.org/RDF/
[9] Semantic Web: Linked Data, http://www.w3.org/standards/semanticweb/data
[10] SPARQL Query Language for RDF http://www.w3.org/TR/rdf-sparql-query/