KUNGL TEKNISKA HÖGSKOLAN

CID-216 ● ISSN 1403-0721 ● Department of Numerical Analysis and Computer Sience ● KTH

Query Management For The Semantic Web

Henrik Eriksson

Master's Thesis, Computer Science Program

CID, CENTRE FOR USER ORIENTED IT DESIGN

KUNGL TEKNISKA HÖGSKOLAN

CID-216 ● ISSN 1403-0721 ● Department of Numerical Analysis and Computer Sience ● KTH

Query Management For The Semantic Web

Henrik Eriksson

Master's Thesis, Computer Science Program

CID, CENTRE FOR USER ORIENTED IT DESIGN

Henrik Eriksson

Query Management For The Semantic Web,
Master's Thesis, Computer Science Program
Report number: CID-216
ISSN number: ISSN 1403 - 0721 (print) 1403 - 073 X (Web/PDF)
Publication date: February 2003

Reports can be ordered from:

CID, Centre for User Oriented IT Design
NADA, Deptartment of Numerical Analysis and Computer Science
KTH (Royal Institute of Technology)
SE- 100 44 Stockhom, Sweden
Telephone: + 46 (0)8 790 91 00
Fax: + 46 (0)8 790 90 99
E-mail: cid@nada.kth.se
URL: http://cid.nada.kth.se

Master's Thesis:
Query Management For The Semantic Web

UPPSALA
UNIVERSITET

Henrik Eriksson

Computer Science Program, 160p
Department of Scientific Computing, Uppsala University
Supervisors:
Matthias Palmér & Ambjörn Naeve, Royal Institute of Technology
Examiner: Eva Pärt-Enander

April 3, 2003

Abstract

This master's thesis has focused on the development of a query management system for the Semantic Web. The latter is a project initiated by the World Wide Web Consortium, aimed at providing better search capabilities to the World Wide Web through an improved metadata framework based on the Resource Description Framework (RDF) standard.

The query management system is mainly designed to interface with the Edutella peer-to-peer network, which is an international initiative to create an RDF-based network that allows highly advanced searches.

The development of the query management system has involved the design of:

- The concept of *template queries*.
- An RDF schema for *forms*.
- The concept of query *workflows*.
- A Java class library for the query management system.
- An example implementation using the class library to add query management capabilities to the Conzilla concept browser.

RDF forms and template queries have been used as a way to provide an easy and flexible interface to complicated queries. Workflows have been introduced to generalize the concept of the query process to ease future extensions, such as e.g. editing.

This work is part of the distributed interactive learning environment that is being developed by the Knowledge Management Research (KMR) group, Centre for User Oriented IT Design (CID), at the Royal Institute of Technology (KTH), Stockholm.

Supervisors have been Matthias Palmér & Ambjörn Naeve, KTH, and examiner Eva Pärt-Enander, Uppsala University.

Contents

1 Abbreviations

In this report the following abbreviations have been used:

API Application Programming Interface

CSS Cascading Style Sheets

HTML HyperText Markup Language

P2P Peer-To-Peer

PICS Platform for Internet Content Selection

RDF-QEL1-5 RDF Query Exchange Language, level 1, 2, ... , 5, respectively

RDF Resource Description Framework

RDFS Resource Description Framework Schema

SGML Standard Generalized Markup Language

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

W3C The World Wide Web Consortium

WWW World Wide Web

XML Extensible Markup Language

2 Introduction

2.1 Administration

This work is part of the distributed interactive learning environment that is being developed by the Knowledge Management Research (KMR) group, Centre for User Oriented IT Design (CID), at the Royal Institute of Technology (KTH), Stockholm.

Supervisors have been Matthias Palmér & Ambjörn Naeve, KTH, and examiner Eva Pärt-Enander, Department of Scientific Computing, Uppsala University.

2.2 The Problem

The Semantic Web is a project initiated by the World Wide Web Consortium that aims to provide a better way of describing resources on the World Wide Web using the Resource Description Framework (RDF) standard (see *Section 3, Background*, for details). RDF-based metadata has significant advantages, such as enabling accurate and arbitrary complex searches.

Edutella is an international initiative that builds on the RDF standard to implement a Semantic Web in the form of a peer-to-peer network (see *Section 3.4.1, Edutella*, for details). Searches on the Edutella network is done by RDF-based queries that are similar to Datalog/Prolog programs.

The Knowledge Management Research (KMR) group at the Centre for User Oriented IT Design (CID) at the Royal Institute of Technology (KTH), Stockholm, has developed a concept browser called Conzilla that supports navigation in an atlas of context maps [3] [4] [5]. A context map is a set of concepts and conceptual relations that are presented together in order to provide the user with a sense of the *context* in which a piece of information is presented. This has general advantages, but in particular it makes it a useful educational tool.

Conzilla has recently been modified to work on RDF-based metadata and it would be useful to be able to interface with Edutella peers, since that would combine the user-centered view of Conzilla with the powerful search capabilities of an RDF metadata network.

The aim of this thesis has been to create a query management system that can be used to add Edutella search capabilities to the Conzilla browser. Since Edutella queries has the potential of extreme complexity, the user interface has a high priority.

To summarize, the aim of this thesis has been the design of:

- A query management system.
- A user interface to queries, with emphasis on simplifying complex queries.
- An interface to the Edutella peer-to-peer network.

2.3 How To Read This Paper

The following is an outline of the sections of this thesis:

Section 3, Background, is a description of the World Wide Web, its limitations, and the improvements to it that are introduced by the Semantic Web initiative. This section tries to illuminate the need of some kind of query management for the advanced query possibilities that follows from a structured metadata system.

Section 4, Analysis, examines the query process, discusses problems with it and how it relates to the process of metadata editing. This section tries to motivate the design decisions made for the query management system described in *Section 5*.

Section 5, Design, describes the design of the query management system.

Section 6, Implementation, provides brief comments about the development environment in which the implementation of the query management system was made.

Section 7, Query Management For The Conzilla Browser, describes the query management system applied to the concept browser Conzilla.

Section 8, Future Perspectives, is a discussion of the possible extensions to the query management system and related issues.

3 Background

3.1 The World Wide Web

The Internet and the World Wide Web (WWW or just "the web") today consists to a large extent of a distributed collection of HyperText Markup Language (HTML) pages. The WWW was developed in the early 1990s at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland, as an effort to ease the sharing of information among their many different computer systems [10]. The basis for the WWW was HTML, which was meant to be a simple format language that structured the information in a document into logical components such as headings, paragraphs, and links [12].

The WWW was a huge success and as it gained a more widespread use the focus shifted from the early content-based view to concerns about the appearance of web pages. HTML version 2 [13] and 3 [14] contained a lot of new layout features which, together with script languages and Java applets, increased the interactive capabilities as well as the visual expressiveness of the web.

3.2 Searching The Web

With the growth of the web the amount of available information has become too large to browse manually. To find information about a particular topic various search engines have to be used. A major problem for these search engines is that in order to find relevant results they have to scan HTML documents to find words or phrases that match keywords used to describe the topic.

The first problem with this approach is that it is difficult to describe a topic with keywords that match the content of all relevant pages to a particular topic without also matching a lot of unrelated pages. A keyword describing a topic might occur in pages unrelated to the topic, and reversely, relevant pages don't necessarily contain the chosen keyword. The task of matching keywords with page content without generating "false positives" is further complicated by the fact that current web pages intermingle content with layout information.

Another even bigger problem with this approach is that the content of some topics isn't text at all. When searching for any type of media files, programs, or other non-textual content, a search engine is totally dependent on the author of the content pages to label the content appropriately.

The problem with intermingled content and layout information can be solved by using a separate layout/style language like e.g. Cascading Style Sheets (CSS) [1]. This leaves the main content page free to concentrate on content while a separate document provides the layout and style directives. HTML version 4 [15] has taken steps in this direction by deprecating a lot of style tags and instead referring to the use of style sheets (e.g. CSS). This has the additional benefit of providing a possibility for different layout information for different audiences, such as voice commands or braille device instructions for blind people.

The problem with non-textual content is trickier. The only really good solution is adequate documentation of the content. This also addresses the problem with search accuracy. Searching information *about* content instead of the content itself is much easier, provided the information about the content is accurate.

3.3 The Semantic Web

3.3.1 Definition

The purpose of the “Semantic Web” is to be an extension of the current web that provides well-defined information about resources. From the official web site, maintained by the World Wide Web Consortium (W3C), the following definition can be found [22]:

> The Semantic Web is the abstract representation of data on the World Wide Web, based on the RDF standards and other standards to be defined. It is being developed by the W3C, in collaboration with a large number of researchers and industrial partners.

By providing a flexible and extendable way of formally describing resources, the Semantic Web enables authors to document their content so that search engines can find it in an accurate and efficient way.

3.3.2 Internet Metadata

Metadata means data about data. In this context it is used to denote the information used to describe web content. Analogous to the benefits of separating layout information from content, there is even greater benefits to be had from separating information about content from the content itself.

An early form of metadata was the use of the `<meta>` tag in HTML. This can be used to give information about a web page such as e.g. the author of the page or a list of keywords. For textual content this information can be described in the content of the page itself, but it is easier for a search engine to know that the text string “John Smith” represents the author of the page when it is labelled within a `<meta>` tag as “author” than when found in the content as “Hi my name is John Smith. I wrote this page!”.

The `<meta>` tag is used by the Platform for Internet Content Selection (PICS) which gives web page authors a way to label and categorize their pages with regard to their content [18]. The main purpose of PICS is to provide a way to label pages so that parents can filter out content unsuitable for children.

The HTML `<meta>` tag approach is a common general way of presenting metadata; as label-value pairs, e.g. “author” = “John Smith”. In order for this to be understandable by search engines, the labels must be well known. If e.g. everyone agreed upon including “author”, “creation date”, and “keywords” as metadata about all web pages then it wouldn’t be of any use to a search engine if someone included their own metadata label “last edited” or even a variant or specialization of the established metadata labels, such as “creator” or “co-author”.

The problem thus becomes, which metadata properties to support? Regardless of how cleverly chosen, the set of metadata properties will always be insufficient for some applications. It’s impossible to anticipate the need of everyone. What’s needed is a way to extend the set of known metadata labels without breaking the backward compatibility. Enter RDF!

3.3.3 RDF

The Resource Description Framework (RDF) was first presented in 1997 as an alternative way of representing information in general and metadata in particular [19] [20]. It is an information description language that addresses many of the problems with metadata presentation.

RDF is:

- Based on simple principles.
- Flexible.
- Extendable without breaking backward compatibility.
- Can be expressed in well-known formats such as XML.

Before elaborating on these points a short introduction to RDF is appropriate. At its core RDF is built up by triples. These triples can be viewed as a graph where each triple is represented by an arrow from one node to another. *Figure 1* shows the simplest RDF graph possible.

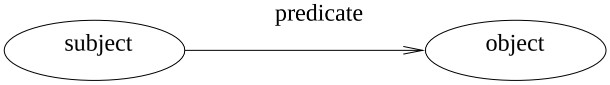


Figure 1: A simple RDF graph.

The semantics of this should be interpreted as *something (the `subject`) has a property (the `predicate`) with the value "`object`".* A concrete example would be the case depicted in *Figure 2*.

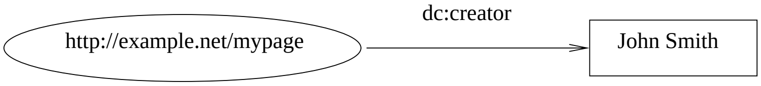


Figure 2: The `dc:creator` property.

This would mean that the page `http://example.net/mypage` was created by `John Smith`. Note that the author `John Smith` is written within a square box as opposed to the page `http://example.net/mypage`, which is written inside an ellipse. This is the RDF graph convention to distinguish between resources (things that are described with URI:s[1]) and literals—sequences of letters and numbers that don't represent unique web locations.

The other thing to note here is the way the property `dc:creator` is written. The "`dc:`" part is an abbreviation for `http://purl.org/dc/elements/1.1/`. RDF only allows resources as properties because literals are not unique and different people could mean different things with e.g. "creator". By using a unique resource as the property you have the opportunity to give the property an exact definition. The object of the RDF property `http://purl.org/dc/elements/1.1/creator` is defined by the Dublin Core Metadata Initiative as

> An entity primarily responsible for making the content of the resource.

[1] The superset of URL:s and URN:s [25] [26].

The Dublin Core Metadata Initiative is an organization that among other things have defined some basic properties that are widely used and recognized [6].

Another reason for distinguishing between resources and literals, besides resources being unique, is that resources, as opposed to literals, can be the subjects of additional triples. There are two reasons why we would want to use a resource instead of a literal to denote the creator of the page `http://example.net/mypage`. The first is that, as mentioned before, string literals are not unique. There is certainly more than one person named "John Smith".

The other reason is that we might want to add additional information about the author, such as e.g. e-mail address and phone number. In *Figure 3*, a unique dedicated resource, `http://example.net/employees/johnsmith`, has been introduced to represent the creator to which more information can be added as new triples. The properties are made up example properties, where "`ex:`" is an abbreviation for `http://example.net/rdf/`.

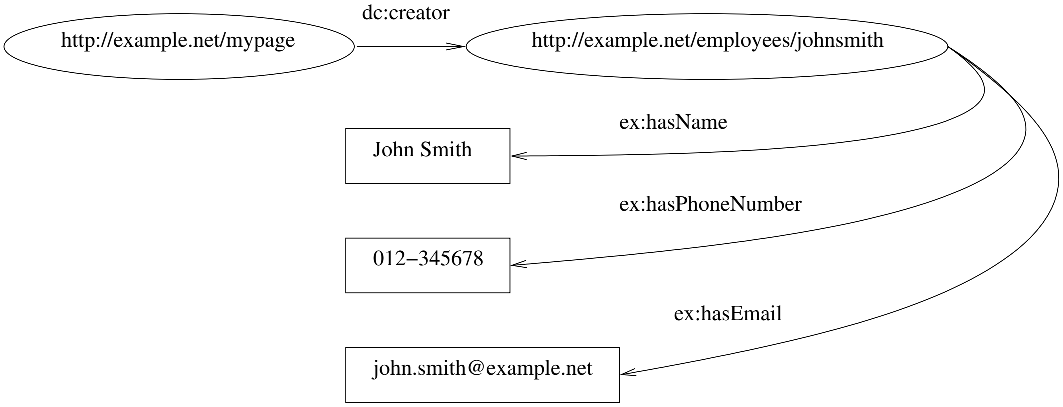


Figure 3: Resources can be both objects and subjects.

3.3.4 RDF Classes

So far we haven't touched upon the real benefits of RDF. What about the ability to extend RDF properties without breaking backward compatibility?

The extensibility of RDF is linked to another feature, classes. Each resource in RDF can belong to one or more classes. This is indicated with the predefined RDF property `rdf:type`, where "`rdf:`" is an abbreviation for `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. In *Figure 4* the resource `http://example.net/employees/johnsmith` is typed to be an instance of the fictive class `ex:Person`, where, again, "`ex:`" is an abbreviation for the fictive base URI `http://example.net/rdf/`.

If we now have an established search engine that knows about the `dc:creator` property and resources of type `ex:Person`, how do we extend this metadata set for use in newer search engines while keeping backward compatibility with the old search engine? As an example extension, consider the case where we want to express that someone is a special type of creator, e.g. a graphic designer, and that somebody is a special type of person, e.g. an employee. What's needed is RDFS.

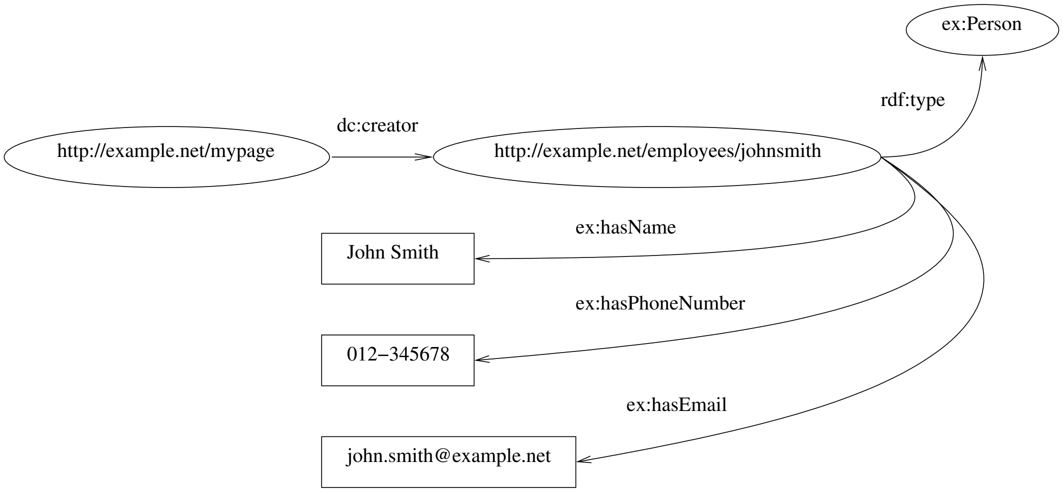


Figure 4: Class membership with the `rdf:type` property.

3.3.5 RDFS

The Resource Description Framework Schema (RDFS) is an extension to RDF that describes how to define RDF vocabularies using RDF itself [21]. It defines, among other things, two important properties, `rdfs:subClassOf` and `rdfs:subPropertyOf`, where "`rdfs:`" is an abbreviation for `http://www.w3.org/2000/01/rdf-schema#`. The semantics of the property `rdfs:subClassOf` is that it denotes a specialization of a more general class. If a resource is of type `T` and `T` is a `rdfs:subClassOf` class `C`, then the resource must also be of type `C`. In analogy, `rdfs:subPropertyOf` denotes a specialization of a more general property.

If we return to our earlier example, we can use RDFS to extend the metadata set without breaking the backward compatibility.

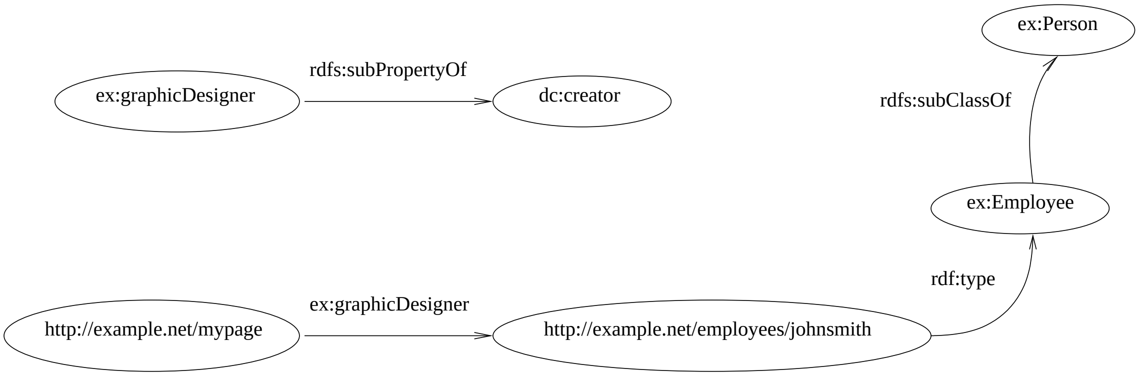


Figure 5: Using RDFS to extend metadata properties.

In *Figure 5*, we have expressed that the individual, John Smith, is the graphic designer (`ex:graphicDesigner`) of the page `http://example.net/mypage` and that he is an `ex:Employee`. If this was the only information available we would certainly have broken the backward compatibility, since the old search engine knows of neither graphic designers nor employees. But in *Figure 5* we have included the information that a graphic designer is a subproperty of the well-known property, `dc:creator`, and that `ex:Employee` is a subclass of `ex:Person`.

By observing that `ex:graphicDesigner` is a subproperty of `dc:creator`, the old search engine knows that `John Smith` is some kind of creator of `http://example.net/mypage`, and although it doesn't know what kind of creator a graphic designer is, this doesn't break backward compatibility—it only means that it can not utilize all information available.

Analogously, the old search engine can deduce that John Smith is an `ex:Person`, since an `ex:Employee` is a subclass of `ex:Person`.

3.4 Searching The Semantic Web

3.4.1 Edutella

The Edutella project [7] is an example of a Semantic Web implementation. It is an international effort to create a metadata infrastructure for peer-to-peer (P2P) networks based on RDF [8]. Although generally applicable, Edutella is mainly (at least initially) aimed towards libraries, universities, and other educational institutions.

The Edutella project is providing an application programming interface (API) and example applications that can act as *producers* and *consumers* of metadata. *Consumers* send queries to the *producers* about what information they are interested in, and the *producers* search their RDF-based metadata and return the relevant results.

The queries sent between the Edutella *consumers* and *producers* are themselves expressed in RDF and their syntax is defined by the Edutella project. The main reason to code the queries in RDF is that they can be stored and treated as any other RDF data by the applications. This means that you could even ask queries about other queries!

3.4.2 Edutella Queries

There are different levels of Edutella queries that differ in their expressiveness and complexity. At the simplest level, the desired information is mimicked and the unknowns are replaced with resources of the type `edu:Variable`, where "`edu:`" is an abbreviation for `http://www.edutella.org/edutella#`.

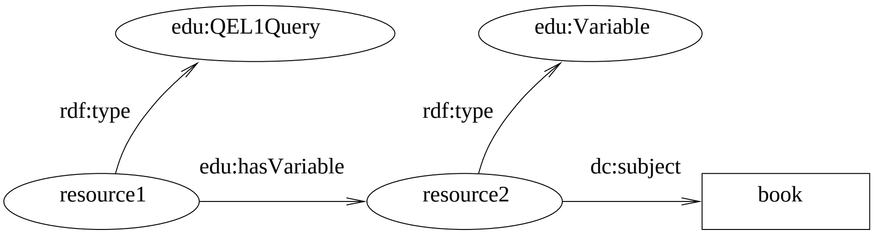


Figure 6: A simple Edutella query.

Figure 6 shows an example of the simplest type of Edutella queries. "QEL1" stands for query exchange language, level 1. This query states, "give me all resources about books". This approach is easy to write and understand, but has limitations. Disjunctions, e.g. "give me all resources about books *or* magazines", are impossible to express and must be separated into two distinct queries.

Edutella query exchange language, level 2 (RDF-QEL2) adds disjunctions, but it's not until we reach level 3 (RDF-QEL3, adds negation) and above that we have a fully expressive query language.

3.4.3 Advanced Edutella Queries

Edutella query exchange language, level 3 (RDF-QEL3) and above are approximately equivalent to Prolog/Datalog. RDF-QEL4 and RDF-QEL5 add different levels of recursion to allow transitive closure.

Before going further, there are two new RDF concepts that need to be defined—sequences and reified triples.

In order to describe ordered sequences, RDF provides a special class, `rdf:Seq`. Resources of this type can have properties `rdf:_1`, `rdf:_2`, etc., which reference an ordered sequence of items. If the actual index of the sequence member doesn't matter, RDFS adds a property, `rdfs:member`, which indicates an unspecified sequence member. As an example of `rdf:Seq` *Figure 7* shows the alphabetically ordered sequence of `apple`, `banana`, and `pear`.

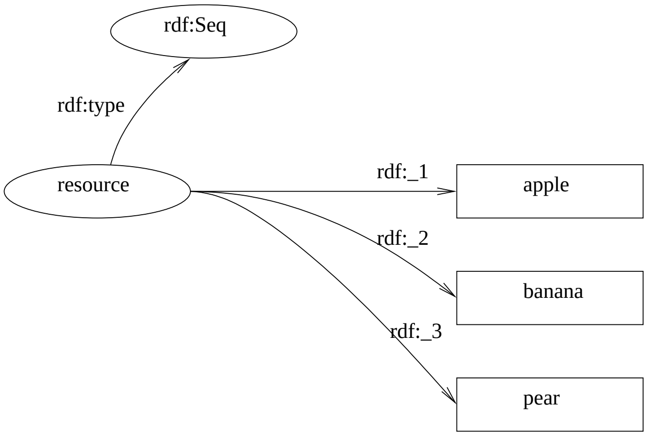


Figure 7: An ordered RDF sequence.

QEL3 and above need to refer to triples in addition to resources and literals. This can only be done indirectly by creating a resource that references all three parts of the triple—subject, predicate, and object. RDF defines this kind of indirect resource to be of the type `rdf:Statement`, and this indirect referencing of the different parts of a triple is called to *reify* the triple. RDF defines the properties of the `rdf:Statement` that refer to the different parts of the reified triple as `rdf:subject`, `rdf:predicate`, and `rdf:object`, respectively. The Edutella `edu:RDFReifiedStatement` class is an extension of the `rdf:Statement` class, and is used in the same way. *Figure 8* shows an RDF triple and its reification.

Edutella RDF-QEL3 queries are based on predicates and rules analogous to Prolog programs, and can express queries that are impossible to manage with simple free-text search engines. As a simple example, imagine that we would like to find out about books written by AI book authors. Note, that we can't just search for "AI" and "books", since the authors

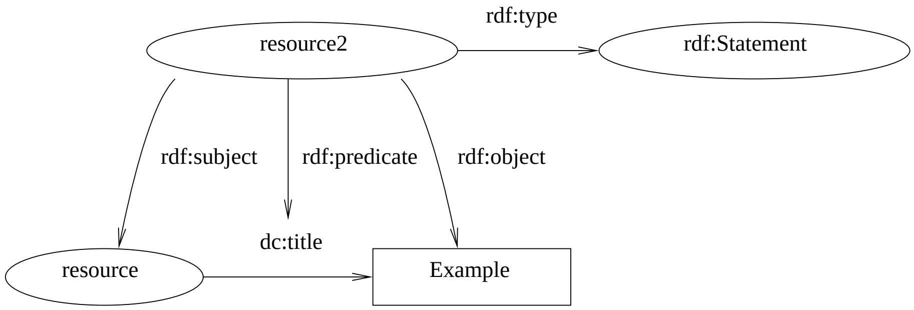


Figure 8: An RDF triple and its reification.

of AI books may well have written books about other topics, and we're probably even more interested in those, judging by the way we phrased our request.

There's no way to phrase the question accurately with a free-text search engine. Instead we would have to separate the search into different parts. We would indeed have to search for "AI" and "books" to find the AI book authors, and then do a book search for each of the authors we find.

In Prolog we would phrase the query by defining one or more rules that logically captures the semantics of our query, together with a query predicate that we can use to ask the question:

```
AIQuery(x):- isBook(x),
             isBook(y),
             hasSubject(y, ‘‘AI’’),
             hasCreator(x, z),
             hasCreator(y, z).
?- AIQuery(x).
```

The Edutella RDF-QEL3 query is exactly analogous to this Prolog program. The RDF-QEL3 query resource has a property `edu:hasQueryLiteral` with a value that corresponds to the Prolog query literal, `AIQuery(x)`. Furthermore, the RDF-QEL3 query resource has a `edu:hasRule` property with a value that corresponds to the definition of the `AIQuery(x)` predicate, and in turn has `edu:hasHead` and `edu:hasBody` properties with values that correspond to the different parts of the Prolog predicate clause.

Although the Edutella RDF-QEL3 query is exactly analogous to the Prolog program, since it is coded in RDF with type definitions and triple reifications, its complete description becomes considerably more complex, as clearly indicated by *Figure 9*.

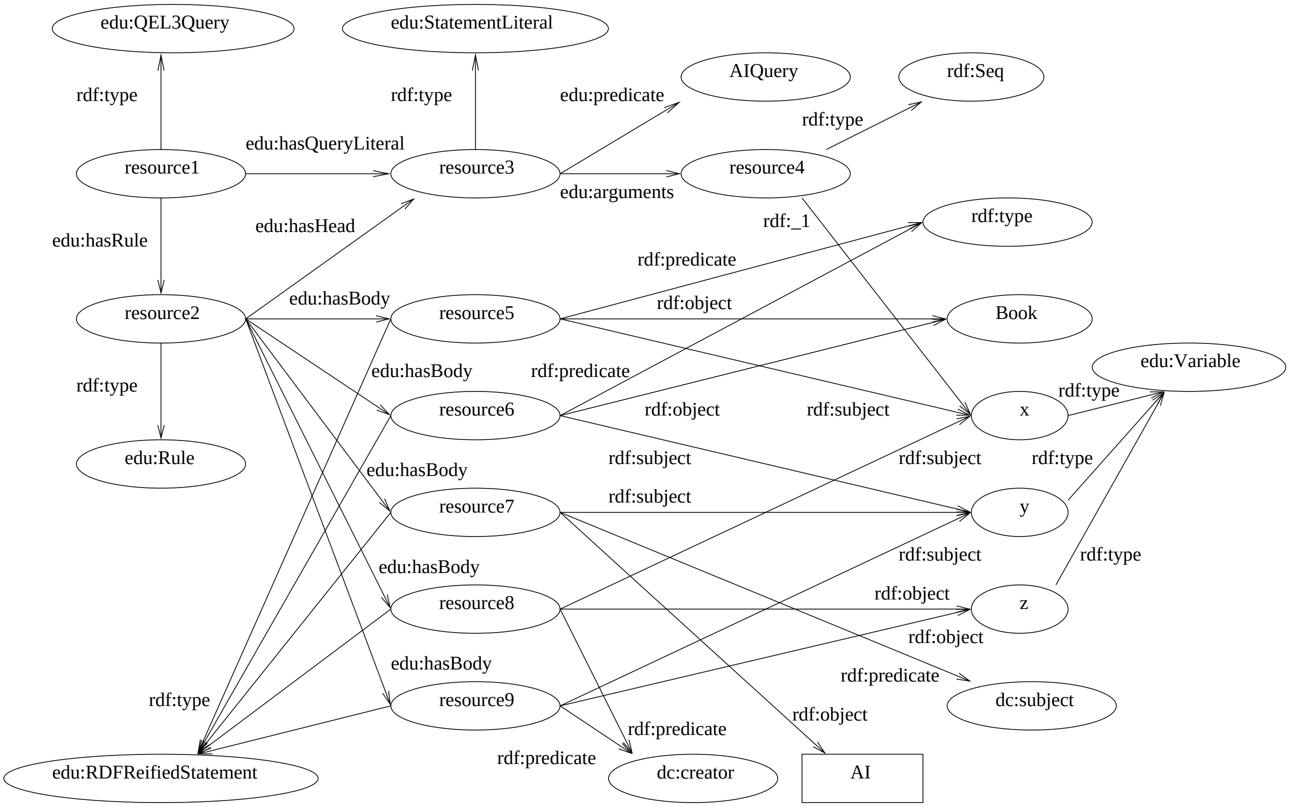


Figure 9: An Edutella RDF-QEL3 query.

4 Analysis

4.1 Query Management

4.1.1 Query Complexity

An Edutella query is essentially a Prolog program (see *Section 3.4.3*). For all but the simplest queries the RDF representation becomes rather complicated (see *Figure 9*).

For a novel user, the task of writing the query in *Figure 9* just to do a search would be a rather daunting task. To the novice, even the RDF graph of the simplest Edutella query, such as e.g. that depicted in *Figure 6*, would not be trivial.

For advanced users, e.g. programmers etc., this would be little more than a nuisance, but if the query management system developed in this thesis project is going to be included in a browser such as Conzilla, which aims to cater for the need of many, there has to be some simplifying mechanism available.

The complexity of the query creation process can be reduced by a good editor. RDF triples describing the type information about a resource can be replaced with special formatting such as showing the resource with a special color or font. Some type information can be automated, e.g. when adding a `edu:hasHead` property, the resource that is the value of that property has to be of the type `edu:StatementLiteral`.

But even if all RDF-specific complexity is removed, the inherent complexity due to the semantics of the query remains. There is no way around this. It's not possible to have a Prolog program without the complexity of a Prolog program.

So, is the only option for reducing the complexity of a query to reduce the inherent semantic complexity of the query, i.e. ask a simpler question? That would be discouraging, because it would mean that all but the most advanced users would miss out on the powerful search possibilities an RDF-based metadata system provides. Fortunately, this is not the case.

4.1.2 Expressiveness And Flexibility

An Edutella query has great *expressiveness* and *flexibility*. These are not the same thing. The expressiveness of the query is, in this context, used as a measure of how complex search conditions we are able to *express* with the query. The different query language levels of the Edutella system are characterized by an increasing expressiveness with higher query language level.

The flexibility of a query is in this context used to describe how fixed or free it is. Before the query is created the flexibility is maximal—it can be used to express most anything. But after it's created the flexibility is zero, its rules and query literals are permanently fixed. This may seem self-evident and the concept of flexibility appear superfluous, but it's crucial for the next topic in the quest for query simplification, template queries.

4.1.3 Template Queries

As mentioned before, creating an Edutella query is in many respects analogous to writing a Prolog program and the execution of the query is analogous to running the program. Although possible for the advanced users, not even programmers would like to write programs from scratch every time they wanted to use them.

Programs normally interact with the user. In the case of a simple Prolog program, the interaction can consist of giving the query literal some appropriate arguments. This approach is clearly less flexible than completely rewriting it for the task at hand, but it doesn't mean that the expressiveness is sacrificed. The ability of a ready-made program to express the solution to a task does not have to be any less than that of a newly written program.

The concept of sacrificing flexibility to reduce complexity can be applied to queries in the form of template queries. A template query is a query where the user gives arguments which replaces parameter variables in the query. This is the same as passing arguments to a query literal in Prolog. By letting the user choose from a set of cleverly designed premade template queries, the difficulty of performing a search is reduced to the design of the user interface to the template queries.

For a given *subset* of tasks, this gives even the novice user access to queries of arbitrary expressiveness. As stated, this comes at the price of flexibility—when a search is needed for which there exists no template query, there is no substitute for creating the query from the ground up. Again, this parallels the situation with programs—when a task needs to be solved for which there exists no program, the program has to be written in order to solve the task.

This doesn't mean that a new query has to be written *completely* from scratch. Since Edutella queries are RDF-based, it's no problem storing them for later reuse, modification, or *combination* into new queries. This is the beauty of letting the Edutella queries be coded in RDF—they *become part of the world upon which they act*.

4.1.4 Forms As Query Interfaces

As seen by e.g. *Figure 9*, the RDF graph of an Edutella query isn't very helpful for understanding what it does. Of course that information can be deduced by analyzing the graph, but if this query was to be the basis for a template query, the nitty gritty details of the query rules wouldn't really be needed by the user. Just as a user of a computer program normally has no interest in the source code of the program, the user of a template query only needs to know what the query does and what parameters are available for changing.

Forms are an intuitive and easy way of interacting with a user. It can be as simple as the single text field for entering search strings that is used by most free-text search engines on the web today. Forms also make it easy to provide explanatory text to the user in the form of headlines and pop-ups.

The problem with forms as a user interface for template queries is that almost all queries will need completely different forms. It's not possible to create a general form that will fit all template queries. This makes it almost hopeless to hardcode the form into the query user interface. The solution is to define a language for describing forms, and as stated before, this is a task to which RDF lends itself very well.

4.1.5 RDF Forms

If forms were coded in RDF they could be stored together with the template queries. With a formal RDF grammar for forms defined, there could be a user interface generator that was able to translate RDF forms into the corresponding form user interface. Forms could then be stored, reused, and combined into new forms like all other objects coded in RDF.

Instead of rewriting and recompiling the code every time a new form or query is needed, a new RDF form and query is specified, which is a significantly less complicated task.

4.2 Workflows

4.2.1 Queries And Editing

Although the most immediate use of queries are for finding resources as specified by users, it's not their only applicability. When editing an existing RDF database, as opposed to creating one from the start, searches have to be made in order to find the resource to be edited along with already existing metadata involving it.

Editing RDF databases is outside the scope of this thesis, but editing is a natural and important extension to this work that will be added at some point. By analyzing the similarities and differences between queries and editing, the query management system can be designed to be more easily extended.

4.2.2 The Query Process

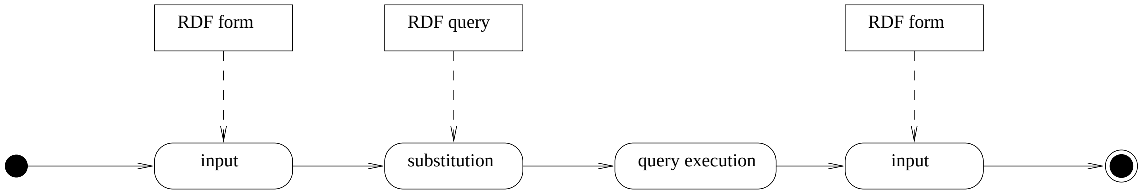


Figure 10: Activity diagram for a template query process.

Figure 10 shows an activity diagram of a typical template query process, assuming RDF forms are used as a basis for the user interface. Rounded boxes mark actions and rectangles objects. Inward arrows to a box means input data for the action, and outgoing arrows means output data. The data objects sent between actions have been omitted for brevity.

The activity diagram attempts to illustrate the different parts of the template query process:

1. A user interface based on an RDF form is used to collect input data from the user, e.g. a search term.

2. The user input data is used as arguments to be substituted for the parameter variables of an RDF template query.

3. The query that is the result of replacing its parameter variables with user arguments is executed.

4. The results of the executed query is displayed via a form that optionally takes new user input for further processing.

The last point might need some comments. The corresponding action is termed `input` rather than e.g. `result display`, because the most general approach would be to let the user decide if the result should be used for further processing, such as e.g. refining the query. In *Section 5* it will be shown that RDF forms are a rather useful basis for displaying results in addition to being used for user interface generation.

4.2.3 The Editing Process

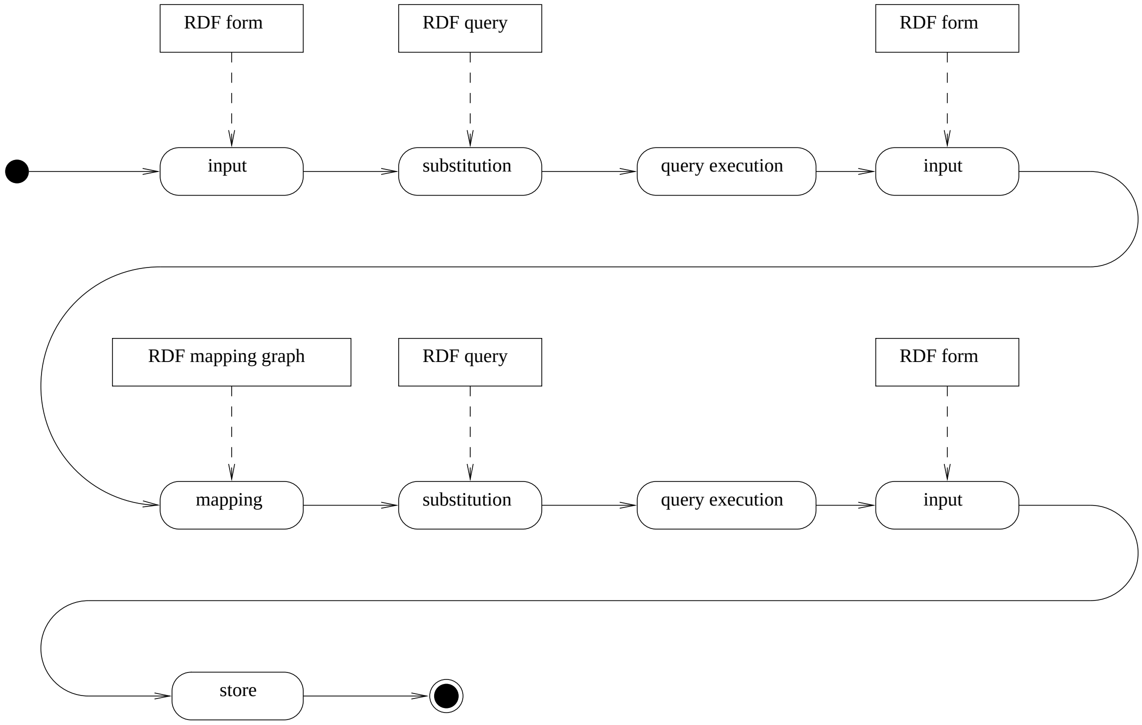


Figure 11: Activity diagram for an RDF edit process.

Figure 11 shows an activity diagram of a typical RDF editing process. Again, as in the query activity diagram, rounded boxes mark actions and rectangles objects. Inward arrows to a box means input data for the action, and outgoing arrows means output data. The data objects sent between actions have been omitted for brevity.

The activity diagram illustrates the different parts of an RDF editing process:

1. A user interface based on an RDF form is used to collect input data from the user about which resource to edit.

2. The user input data is used as arguments to be substituted for the parameter variables of an RDF template query that is designed to find resources to be edited.

3. The query that is the result of replacing its parameter variables with user arguments is executed.

4. The resulting resource candidates that were found when executing the query is displayed via a form so that the user can choose the resource to be edited.

5. A mapping graph is used to map the chosen resource to a parameter variable in a new template query which is designed to find metadata about the resource.

6. The chosen resource is used as a substitution argument for the mapped parameter variable in the query.

7. The substituted query is executed.

8. The found metadata involving the resource is displayed via a form so that the user can edit it.

9. The edited metadata is stored.

The reason for the variable mapping in step 5 is to specify which parameter variable in the new query the resulting resource should be substituted for. This information can not be hardcoded, since it would lock the forms and queries involved. By coding this information in an RDF graph, it can be stored and used in the same flexible way as the RDF forms and RDF queries.

4.2.4 Workflows

By comparing *Figure 10* and *Figure 11* two things can be noted:

- Queries are a subset of RDF editing. The first half (step 1-4) of the editing process is a query.

- By looking at the subparts of the two processes, it's clear that the overlap is even greater. The `input`, `substitution`, and `query execution` steps occur in the query process and several times in the editing process.

If the query management system is designed with this in mind, it will be easier to extend to include editing, with a minimum of duplicated effort. If the parts of the query process system is modularly designed, they can be reused when designing the editing extension.

The notion of a workflow could itself be abstracted into an independent entity. With cleverly chosen building blocks based on the process parts in *Figure 10* and *Figure 11*, there wouldn't be any need to distinguish between a query process and an editing process. All processes would be some kind of workflows with elements of editing, querying, and perhaps additional future components when the need arises.

One could even take this further and design a grammar for coding the workflows in RDF. With a workflow interpretation engine that translates RDF workflows into processes, designing a new process would consist of creating a new RDF workflow graph. This would give workflows the benefits of everything else coded in RDF—storage, reuse, recombination into new workflows. It would be the RDF equivalent of a scripting or macro language.

In this work the first steps towards this goal will be taken, but the full implementation is outside the scope of the thesis.

5 Design

5.1 The Query Management System

5.1.1 Design Goals

The aim of this project is to create a query management system that is:

- Easy to use.
- Flexible.
- Extendable.

The powerful search capabilities that come with RDF-based metadata mean that it's possible to state potentially very complex queries. A very important part of the overall usefulness of the query manager is the ability to provide some simplifying mechanism for complex queries.

Flexibility means making as few assumptions as possible. As much implementational detail as possible should be kept unexposed in order not to limit alternative future solutions. This is a general rule of thumb that applies to data structures as well as algorithms.

Regarding RDF, queries are needed in other tasks than pure searching, such as e.g. editing. The class library used to build this query management system will be extended to include editing at some point. Knowing this, it is desirable to make the management system extensible to avoid having to rewrite a lot of code whenever new features are added.

5.1.2 Query Management Overview

There are three main concepts that have guided the development of this query management system:

- Template queries.
- RDF forms.
- Workflows.

Template queries is a way of reducing query complexity without also reducing the possibility for novel users to perform advanced queries. Of course this comes at the price of being limited to execute only the types of queries for which templates exist.

RDF forms is a supplement to template queries. Due to the complexity of RDF graphs for advanced queries, there is a need for a simpler user interface. Forms are a natural choice, but since the queries can be very different there would have to be a different form for every query. By coding the forms in RDF and having an RDF form interpreter that translates RDF forms to user interfaces, hardcoding forms can be avoided.

Workflows are a generalization of the query process in order to make it more extendable. By defining a workflow as a sequence of subparts of the query process, extending workflows to include e.g. editing means only to add the subparts of the editing process that's not already included in the query process.

5.2 RDF Forms

5.2.1 Overview

RDF forms are forms coded in RDF that can be translated into form user interfaces. The rationale for using forms as query user interfaces has been detailed in *Section 4.1*.

In short, an RDF form is used as a description to a form user interface for a specific template query. After presenting the parameter variables of the query to the user, the user input is collected and substituted for the parameter variables of the query, which is then ready to be executed.

5.2.2 Template Queries

The reason template queries are included here is that RDF forms and template queries are tightly coupled.

Template queries are queries where at least one variable acts as a parameter variable. By substituting the parameter variables with argument values, the template query is converted into a query that is ready for execution.

In order to define a template query, its parameter variables need to be specified. This could be done by defining a new parameter variable type that extends the original variable type. Subclasses of the parameter variable type could give further information about the type of values that could be substituted for them.

This isn't necessary though. As seen in *Section 4.1*, template queries need associated RDF forms. An RDF form needs to keep references to the parameter variables its various parts refer to. Since the information about the available parameter variables of a template query is stored in its associated RDF form, there is no need to store that information in the query itself. This makes sense, since the information about the parameter variables and the types of values that constitute allowable substitutions is only relevant when getting the information from the user, i.e. when using the form.

This has the consequence that there is no difference between a template query and an ordinary query. Template queries are simply the queries associated with RDF forms, and parameter variables are the query variables referenced by the different parts of a form. Parameter variables, for which the user doesn't specify values, are simply left as variables in the query. The more variables in a query for which values are substituted, the more specific the result of executing that query will be.

5.2.3 RDF Form Classes

The RDF form classes that are described here are the resource types that are specific to RDF forms. General RDF classes such as `rdf:Alt` (see *Section 5.2.5*) etc. are omitted although they are used in RDF forms.

In the RDF form class and property names, "`kmr:`" is an abbreviation for the URI base `http://kmr.nada.kth.se/rdf/form#`.

`kmr:Form`

This is the main RDF form type. The root resource of an RDF form is an instance of this class. This class is a subclass of `rdf:Seq` and has zero or more form item children.

`kmr:FormItem`

This is the base class of all form items. The three form item types are specializations of this class.

`kmr:TextFormItem`

This form item is a specialization of `kmr:FormItem` that represents a free-text input field.

`kmr:ChoiceFormItem`

This form item is a specialization of `kmr:FormItem` that represents a choice combo box with a number of alternative choices.

`kmr:GroupFormItem`

This form item is a specialization of `kmr:FormItem` that represents a group of other form items. It is a subclass of `rdf:Seq` and can have zero or more form item children.

`kmr:Query`

This is a superclass of all types of queries that can be referenced by a form. An example subclass is `edu:QEL3Query`.

`kmr:Variable`

This is a superclass of all types of variables that can be referenced by a form item. An example subclass is `edu:Variable`.

`kmr:Style`

This is a class whose members represents style and layout information about how to display a form. This class is only included for future use, when RDF form style information is implemented.

5.2.4 RDF Form Properties

The RDF form properties that are described here are the properties that are specific to RDF forms. General RDF properties such as `rdf:type` etc. are omitted although they are used in RDF forms.

As before, “`kmr:`” is an abbreviation for `http://kmr.nada.kth.se/rdf/form#`.

`kmr:query`

This is a property whose value is the RDF template query that the form is intended to collect user input data about.

`kmr:variable`

This is a property whose value is a variable, belonging to an RDF template query, that a form item is intended to collect user input data about.

`kmr:minMultiplicity`

This is a property whose value determines the minimum number of copies of a form item that should be displayed. If there is no minimum multiplicity property this defaults to 1. This property is only included for future use, when editing is added.

`kmr:maxMultiplicity`

This is a property whose value determines the upward bound on the number of copies of a form item that are allowed to be displayed. If there is no maximum multiplicity property, there is no upward bound. This property is only included for future use, when editing is added.

`kmr:choices`

This is a property whose value represents a number of choices that are available for a `kmr:ChoiceFormItem`.

`kmr:style`

This is a property whose value gives style and layout information about how to display the form. This property is only included for future use when RDF form style information is implemented.

5.2.5 RDF Form Example

In *Figure 12* an example RDF form for finding employees is shown. All titles have multiple language translations. When giving different translations of a string in RDF, it's common to use a resource of the type `rdf:Alt`, which is similar to `rdf:Seq`—see *Figure 7*, but which specifies alternatives instead of a sequence. The language of the translation alternatives is given as part of the translated string, in this example `en:` for English, and `sv:` for Swedish.

The form resource has only one child, a group form item, which in turn has two form item children, a text form item and a choice form item. The text form item represents a free-text input field to fill in the name of the employee and the choice form item represents a choice of specifying a limited number of employee roles, `Writer` and `Artist`.

There can be many reasons for wanting to provide the user with a choice between alternatives instead of a free-text input field. Perhaps the designer of the form (and associated query) knows that there only exists two employee roles, or that it's only these two roles that are interesting in this context.

Figure 13 shows a possible rendering of the RDF form example from *Figure 12* into a form user interface. Here the English title translations have been used. The choice box rendered from the choice form item is shown in the "drop-down" state for clarity.

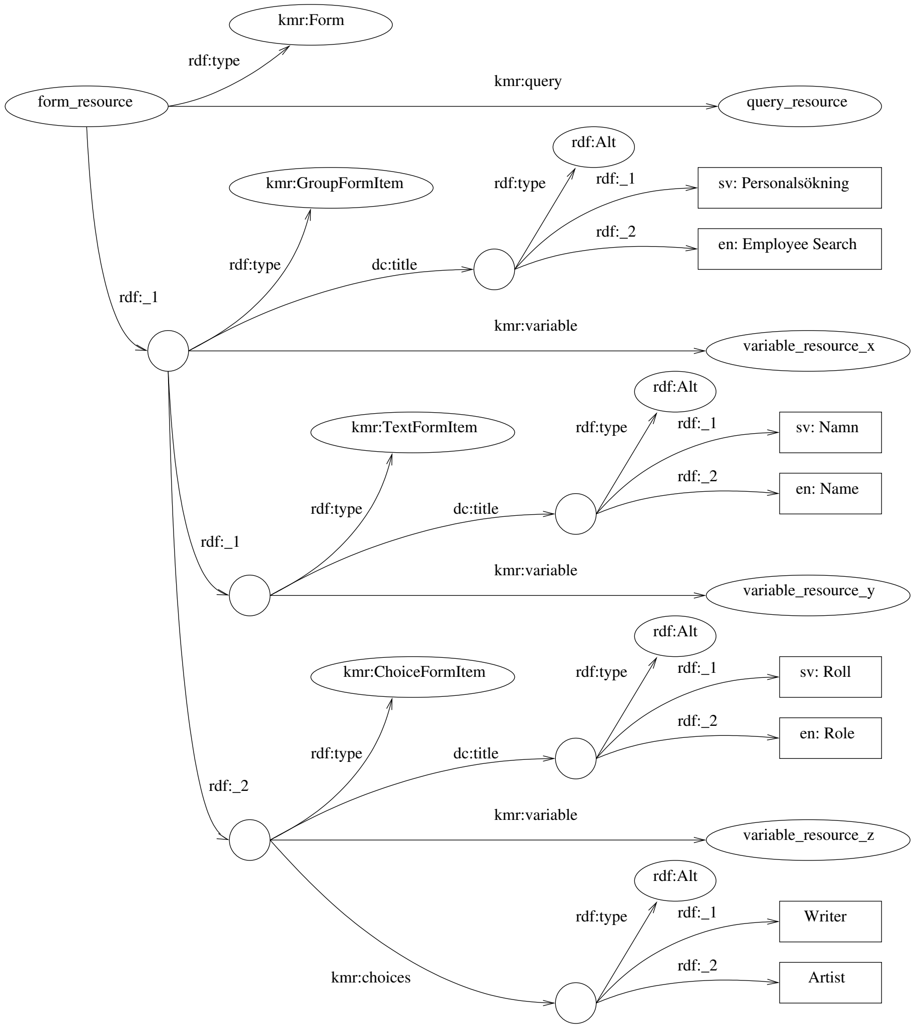


Figure 12: An example of an RDF form.

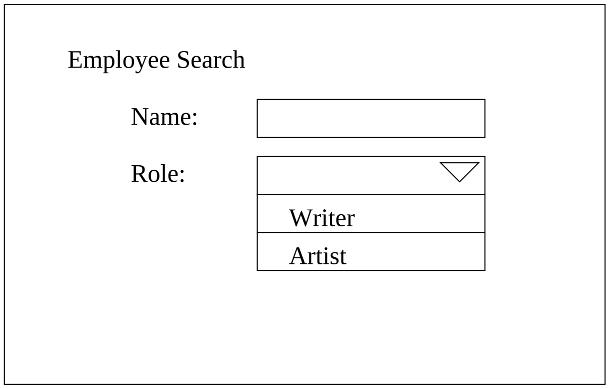


Figure 13: An example of a form user interface rendered from an RDF form.

5.2.6 RDF Schema For Forms

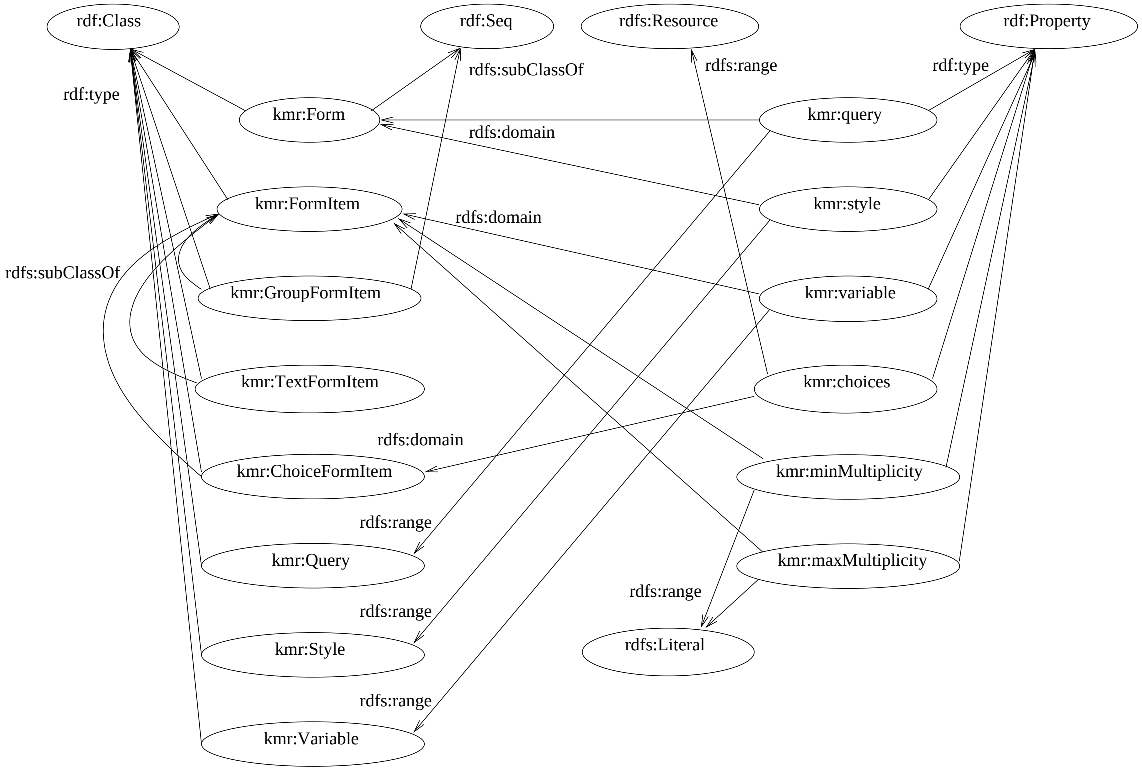


Figure 14: RDF schema for forms.

As mentioned in *Section 3.3.5*, RDFS provides a way to formally describe RDF vocabularies using RDF itself. *Figure 14* shows the RDF schema used to define the vocabulary for forms. There are several RDF classes and properties available for describing an RDF vocabulary.

- `rdfs:Resource` is the set of all RDF resources. All things described by RDF is a member of this class.
- `rdfs:Class` is the set of all RDF classes. All RDF classes are instances of this class.
- `rdf:Property` is the set of all RDF resources that are properties.
- `rdfs:Literal` is the set of all literal values, such as strings.
- `rdf:type` is a property that says that a resource is a member of a class.
- `rdfs:domain` is a property that specifies a class whose members an RDF property can be applied to.
- `rdfs:range` is a property that specifies a class whose members an RDF property can have as values.

The last two properties are used to describe properties. rdfs:domain specifies what kind of resources a property can be applied to, and rdfs:range specifies what kind of values a property can have. As an example we see in *Figure 14* that instances of the kmr:FormItem class can have the property kmr:variable and that the value of that property is of the type kmr:Variable.

The kmr:Query and kmr:Variable classes are collective superclasses of all concrete query and variable classes, respectively. As an example, edu:QEL3Query is an rdfs:subClassOf kmr:Query and edu:Variable is an rdfs:subClassOf kmr:Variable.

There is one thing about the RDF form vocabulary that can't be expressed with RDFS. There is no way to specify typed containers. Hence, the fact that the classes kmr:Form and kmr:GroupFormItem are sequences whose items are instances of the class kmr:FormItem is not something that can be formally stated in RDFS, but has to be added as a comment.

5.3 Workflows

5.3.1 Overview

Workflows are a generalization of the query process to make it more extendable. As already stated in *Section 4.2*, the implementation of a full fledged RDF workflow macro language is outside the scope of this thesis. The concept of workflow has nevertheless been important to the development of the query management system for two reasons, both of which are related to extensibility.

The first is that an RDF workflow macro language is a powerful concept that will be implemented at some point, and it's desirable to create the query management system so that it can easily be extended to incorporate RDF workflows.

The second is that the idea of workflows gives a good suggestion about suitable modularization of the query management system. As discussed in *Section 4.2*, separating the implementation of the parts responsible for the different steps of the query process and making them independent, makes the system generally more extensible. This will in particular affect future integration of editing, since many of the query processing steps are shared with the editing process.

5.3.2 Workflow Components

The workflow components are objects sent between the workflow activities as input or output data. They are all implemented as interfaces in the Java class library of the query management system. The names of the workflow components given here are those of their Java interface counterpart.

`VariableBindingSet`

The `VariableBindingSet` represents the result of a query. It consists of variable bindings, each of which represents a variable and a value bound to it. Although the concrete representation of the variable bindings of a `VariableBindingSet` is unspecified, the variable bindings can be logically grouped into *result tuples*. A result tuple represents a possible solution to a query and consists of the complete set of variables with a single value bound to each of them. The variable bindings of a `VariableBindingSet` can represent more than one result tuple because a query can have more than one solution.

A `VariableBindingSet` is not only used to store the result of a query, but is a general result storage structure for values bound to variables.

`FormModel`

The `FormModel` is an abstract representation of the information needed to create a form user interface.

`QueryModel`

The `QueryModel` is an abstract representation of a query.

5.3.3 Workflow Actions

The workflow actions are the fundamental building blocks of the workflow that are depicted in *Figure 10* and *Figure 11*. Each take one or more workflow components as input and produces one workflow component as output.

They are all implemented as methods of the `WorkFlowManager` interface in the query management Java class library. The names of the workflow actions given here are those of their interface method counterparts.

`substituteVariables`

Input	`VariableBindingSet`
	`QueryModel`
Output	`QueryModel`

The `substituteVariables` workflow action transforms a template query, represented by a `QueryModel`, into an executable query, represented by another `QueryModel`. The input `VariableBindingSet` is assumed to contain a unique variable binding for each of the parameter variables of the template `QueryModel` that is to be substituted.

`executeQuery`

Input	`QueryModel`
Output	`VariableBindingSet`

The `executeQuery` workflow action executes a query represented by a `QueryModel` and returns the result as a `VariableBindingSet`. The query is normally executed by sending it over a network, such as the Edutella peer-to-peer network, to a query engine that returns the results. The results don't have to be returned at the same time. The query engine might very well return result tuples as it finds them. This means that the `VariableBindingSet` is continually updated as new results are reported.

`input`

Input	`FormModel`
Output	`VariableBindingSet`

The `input` workflow action takes a `FormModel` and creates a form user interface. The returned `VariableBindingSet` is used to store the user input data, but it doesn't contain valid data until the user is done, at which point a `message` workflow action notifies the workflow manager.

`message`

Input	`VariableBindingSet`
Output	-

This is the only workflow action that isn't represented in *Figure 10* or *Figure 11*. The `message` workflow action is used to notify the workflow manager that the specified `VariableBindingSet` now has valid data (see the `input` workflow action).

5.4 Program Structure

5.4.1 Overview

The purpose of this section is to give a sense of the overall structure of the most important classes and interfaces that comprise the query management system. The emphasis is on the relations between the classes and the interfaces rather than on details regarding individual classes or interfaces.

The program structure description is divided into three parts based on the purpose and responsibilities of the concerned classes and interfaces—the workflow-, query-, and form-specific parts. This distinction is not absolute. Where there have been overlaps, the involved classes and interfaces have been included in all relevant parts.

The program structure has been modeled using the Unified Modeling Language (UML) [24]. Intersecting lines can be shown in different ways when drawing paths in UML diagrams. In the UML diagrams used here all line crossing indicates connection between the lines.

5.4.2 The Workflow-Specific Classes And Interfaces

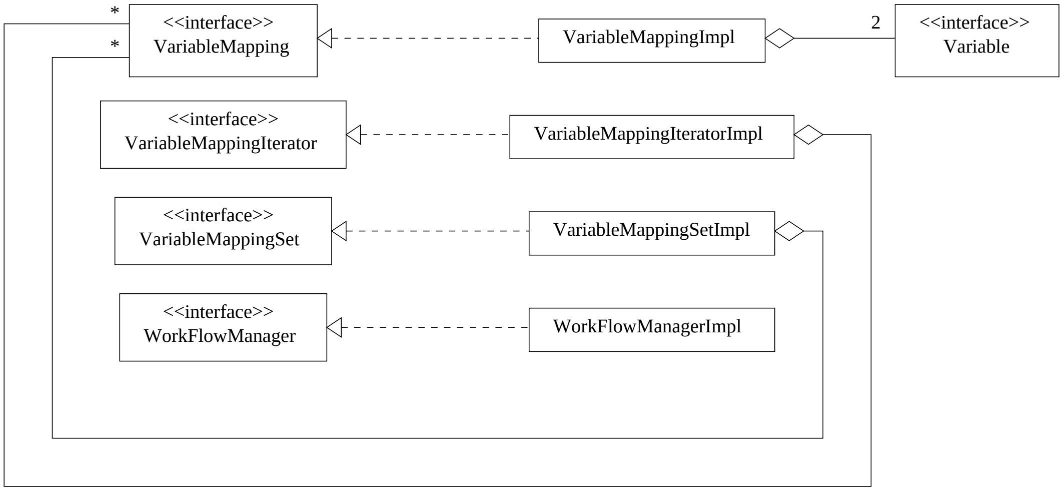


Figure 15: UML diagram for workflow-related classes and interfaces.

All classes and interfaces of the query management system is at least indirectly involved with the workflow of executing a query. *Figure 15* shows the classes that are more directly or exclusively involved with the workflow process. Here the classes and interfaces involved in the editing specific `mapping` workflow action (see *Figure 11*) have been included as well.

RDF encoded workflows that are being interpreted into a macro language are outside the scope of this work. The only workflow that is implemented is the workflow of a query process, which is hardcoded into the `QueryWorkFlowManagerImpl` class.

5.4.3 The Query-Specific Classes And Interfaces

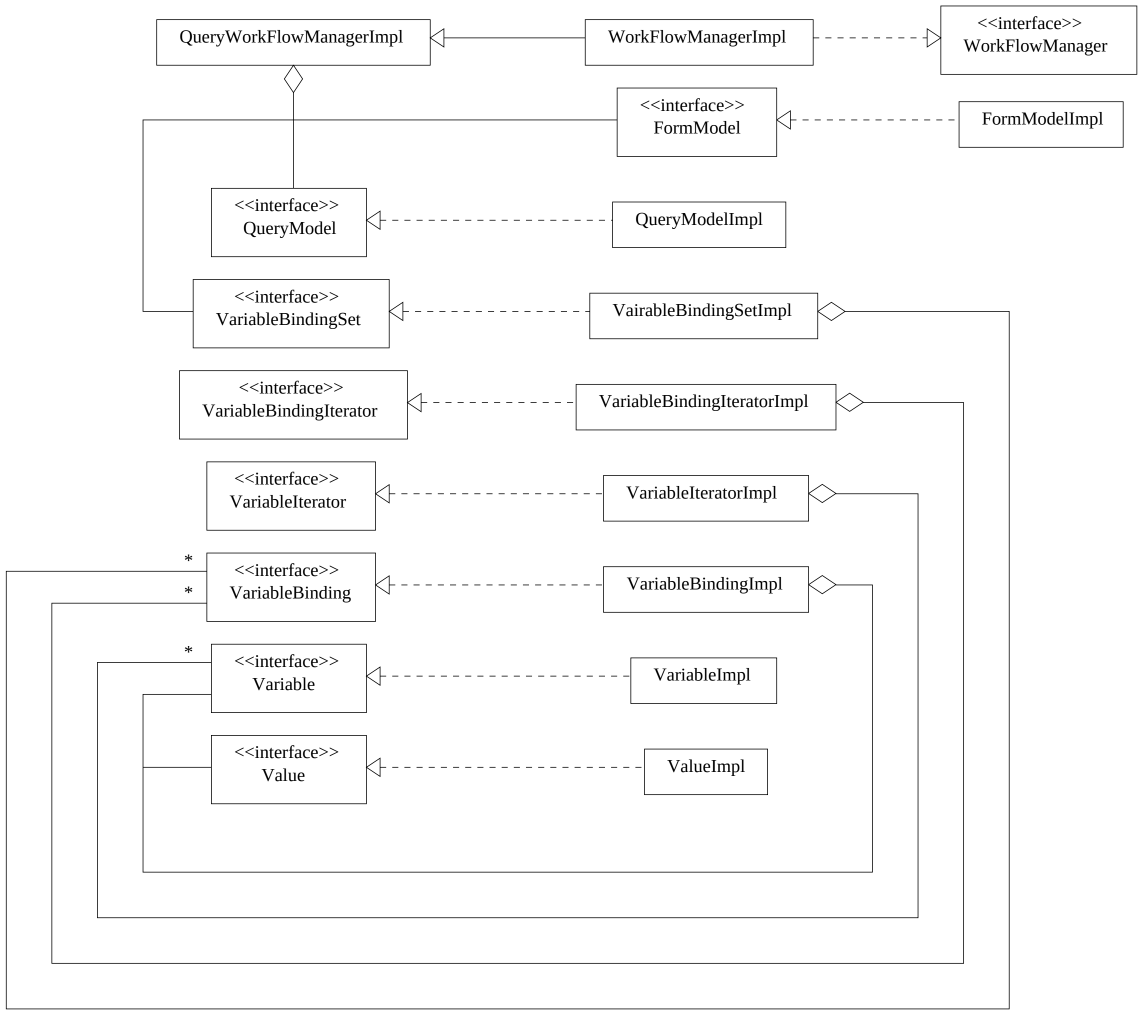


Figure 16: UML diagram for query-related classes and interfaces.

Figure 16 shows the relationships between the classes and the interfaces that are more directly involved with the manipulation and execution of queries. A more detailed description of some of the most important interfaces can be found in *Section 5.3*.

5.4.4 The Form-Specific Classes And Interfaces

Figure 17 shows the relationships between the classes and the interfaces that are involved with the generation of form user interfaces. A more detailed description of some of the most important interfaces can be found in *Section 5.3*.

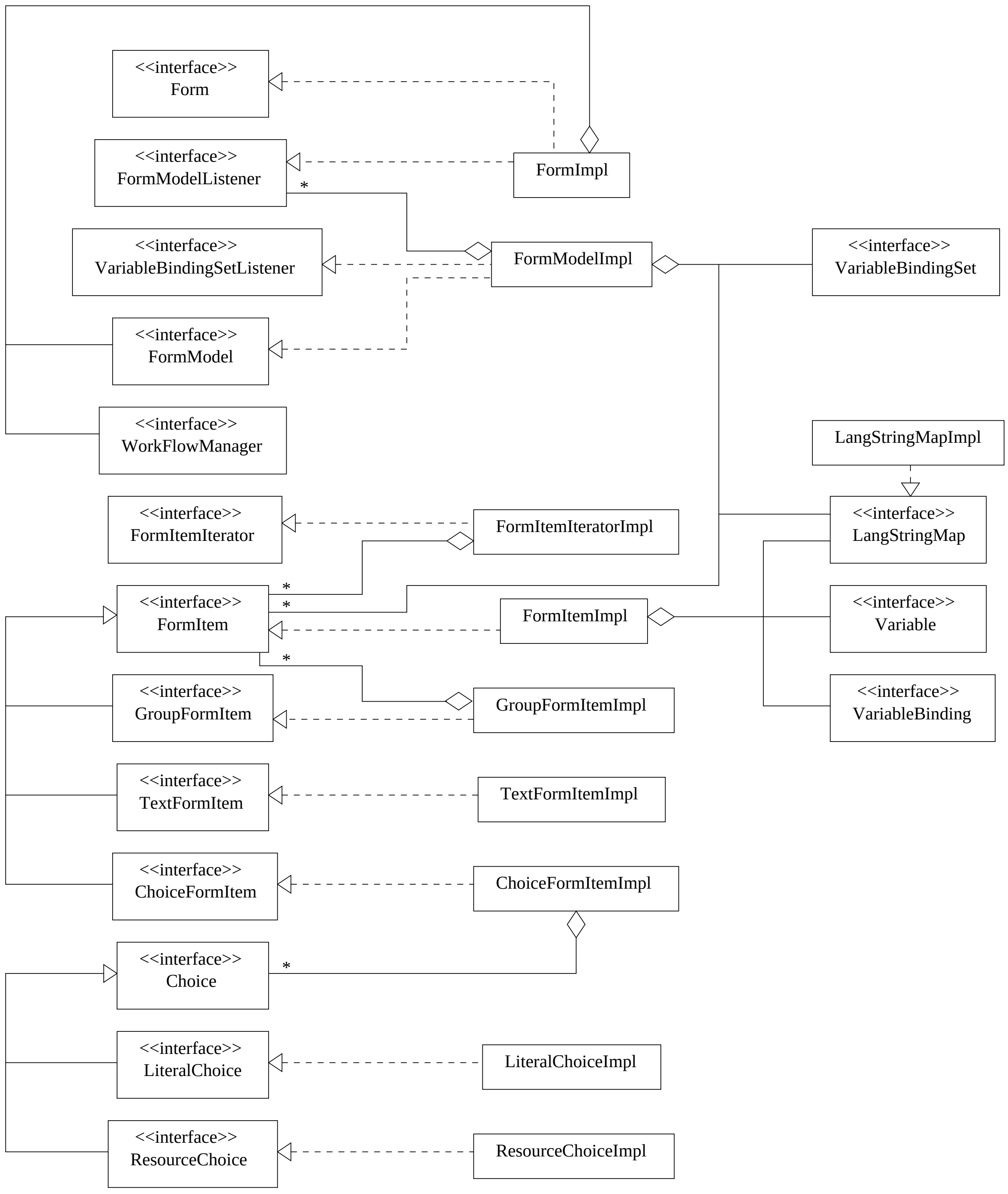


Figure 17: UML diagram for form-related classes and interfaces.

6 Implementation

6.1 Development Environment

6.1.1 Open Source

The source code of the implementation of this thesis work is released under the GNU General Public License, as published by the Free Software Foundation [9]. This guarantees that the source code will remain open and accessible to others.

6.1.2 Java

The implementation of this thesis work has been done using the Java language. There are several reasons for this.

- There are free compilers for Java.
- The Java Virtual Machine, on which Java programs run, exists for most architectures and operating systems, which makes Java programs very portable.
- The Java API is feature rich and easy to use, which means that an integrated development environment isn't necessary.
- The Java language is purely object oriented, promoting good program structure.
- Java has a built-in garbage collector, which leads to easy memory management.
- Java hides pointers and thereby avoids common programming errors.
- The Edutella API and the Jena API are written in Java.

6.1.3 Ant

Compilation and re-compilation of multiple source files quickly becomes tedious when done manually. In this work, Apache Ant has been used as a compilation manager. Apache Ant is an open source Java-based build tool, similar to the Unix `make` utility [2].

The advantage of Ant over `make` is that Ant uses standard XML files for specification of file dependencies instead of the error prone Makefile of `make`, and that Ant can be used on almost all platforms due to it being written in Java.

6.2 External API:s

6.2.1 Edutella

The Edutella API is an open source Java class library containing code to help setting up Edutella provider and consumer peers [7]. In addition it contain classes for manipulating Edutella RDF queries.

The Edutella API was used in this work for setting up a simple Edutella consumer peer and for creating Edutella queries to send via the consumer peer over the Edutella network.

6.2.2 Jena

Jena is a general purpose open source Java API for manipulating RDF data that is being developed by the Hewlett-Packard Company [17].

Jena has been used in this work for low level manipulation of RDF data models, such as e.g. getting and setting properties of RDF resources.

6.3 The Query Management API

6.3.1 Overview

The query management API is divided between four Java packages; `form`, `query`, `workflow`, and `internationalization`. These packages contain no classes, only interfaces. Instead the class implementations are contained in an `impl` subpackage of each of the four packages, respectively.

6.3.2 Java Packages

The naming of the Java packages follows the convention of constructing the root of the package name by reversing the URL of the company or organization responsible for the development of the package.

Hence, the prefix of all packages is `se.kth.nada.kmr.shame`, where the acronym, SHAME, is the name of an ongoing project at KMR to develop a Standardized Hyper Adaptable Metadata Editor, in which this work is intended to be incorporated.

`se.kth.nada.kmr.shame.form`

This is the package responsible for forms. It includes Java interfaces detailing the behavior of both abstract representations of forms and actual form user interfaces.

`se.kth.nada.kmr.shame.form.impl.vocabulary`

This package is responsible for a Jena implementation of the form RDF vocabulary definition.

`se.kth.nada.kmr.shame.internationalization`

This package contains Java interfaces describing multi-lingual strings, i.e. strings for which multiple translations exists.

`se.kth.nada.kmr.shame.query`

This package is responsible for the abstract representation and behavior of queries and variable binding sets.

`se.kth.nada.kmr.shame.workflow`

This package defines the behavior of workflows. It also includes interfaces for variable mappings, which will be needed when the query system is extended to include editing.

7 Query Management For The Conzilla Browser

7.1 Conzilla

Conzilla is a concept browser developed by the Knowledge Management Research (KMR) group at the Centre for User Oriented IT Design (CID) at the Royal Institute of Technology (KTH), Stockholm [3] [4] [5]. It supports navigation in an atlas of context maps, which are sets of concepts and conceptual relations that are presented together in order to give the user a sense of the *context* in which a piece of information is viewed.

A part of this thesis work has been to incorporate a query management system into the Conzilla browser and provide the capability of executing queries on an Edutella peer-to-peer network.

7.2 Example Query Execution

7.2.1 The Edutella Provider

In order to demonstrate the query management of Conzilla, an Edutella network was set up with a provider peer configured with the RDF knowledge base depicted in *Figure 18.* As before, "`ex:`" is an abbreviation for the example base URI `http://example.net/rdf/`.

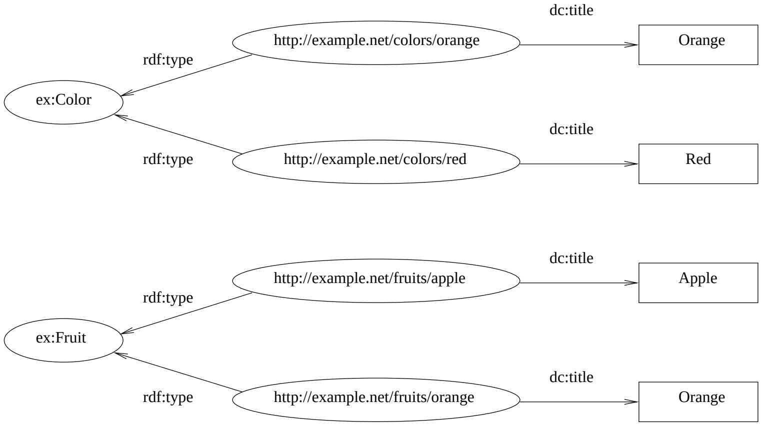

dc:title
http://example.net/colors/orange
Orange
rdf:type
ex:Color
dc:title
rdf:type
http://example.net/colors/red
Red
dc:title
http://example.net/fruits/apple
Apple
rdf:type
ex:Fruit
dc:title
rdf:type
http://example.net/fruits/orange
Orange

Figure 18: RDF knowledge base for the Edutella provider.

7.2.2 An Example Query

Conzilla provides the possibility to draw and display RDF graphs. *Figure 19* shows an Edutella RDF template query with an accompanying RDF form. All type information is hidden and resources are shown as ellipses with their titles as labels.

The complete RDF graphs for the query and the form are displayed in *Figures 20* and *21*, respectively.

The query states, *find all resources that have a type and a title.* When an Edutella query has two query literals, as is the case here, it is interpreted as a conjunction—both query literals have to be satisfied for the query to be satisfied.

The accompanying form is designed to let the user specify the type and/or the title of the resource to search for. For demonstration purposes two different input methods were chosen, free-text input for the title and choice alternatives for the type.

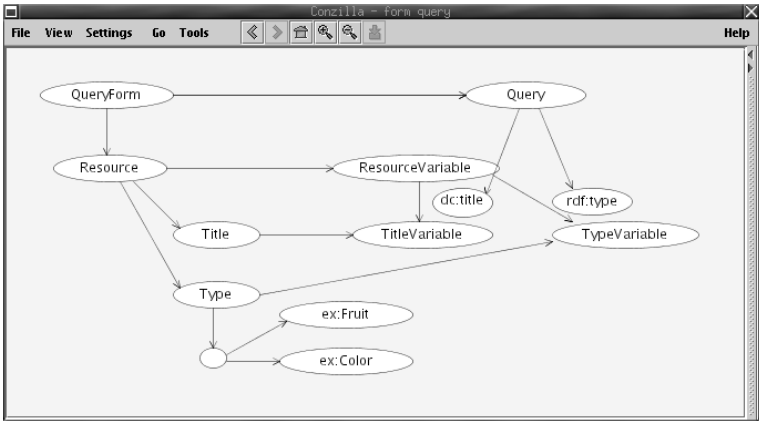


Figure 19: An Edutella query with an accompanying RDF form in Conzilla.

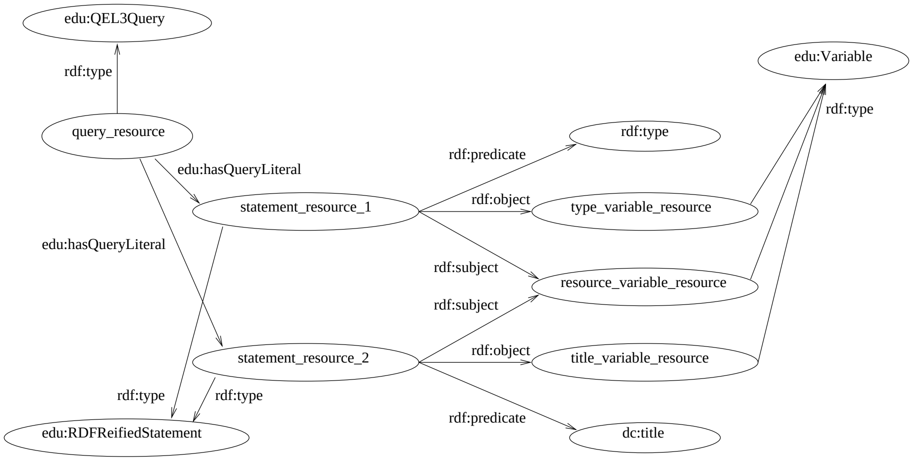


Figure 20: An example Edutella query.

7.2.3 Query Execution

The query interface is started from a pop-up that appears when right clicking on the query resource. If a valid RDF form is linked to the query resource, a form is generated and launched in a separate window, as shown in *Figure 22*.

As a future extension, one could couple hidden forms and template queries to concepts in a Conzilla context map, in order to provide a default search interface for finding content about that concept.

The query is now ready to be executed by pressing the `OK` button. If no information is entered, the query is executed unaltered, leading to the result form shown in *Figure 23*.

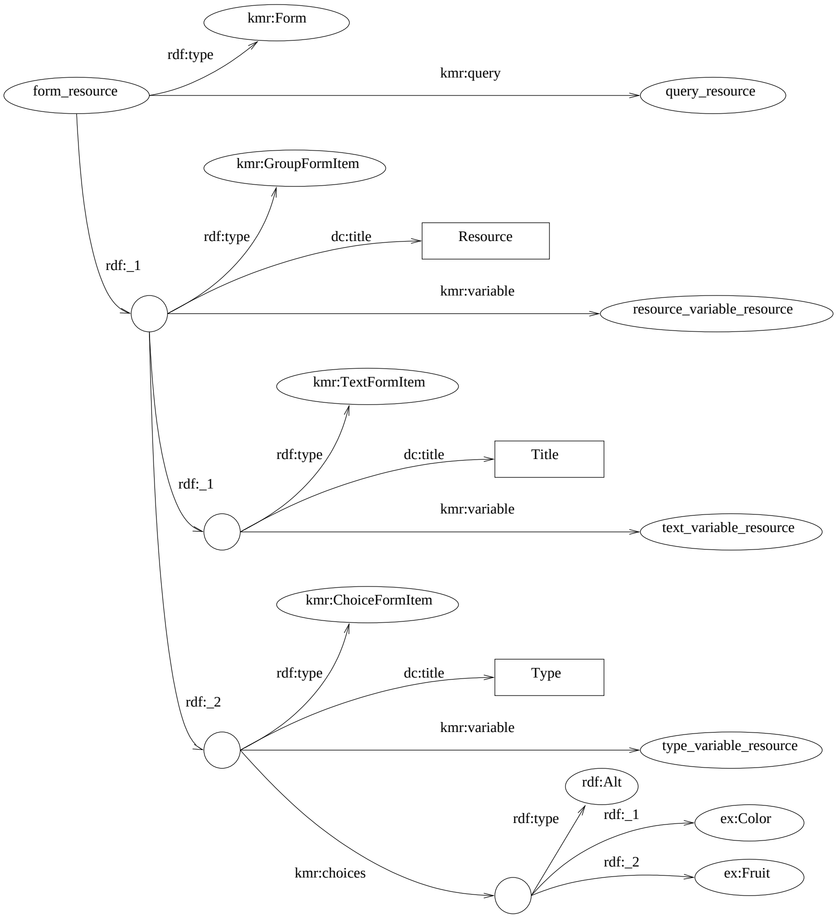
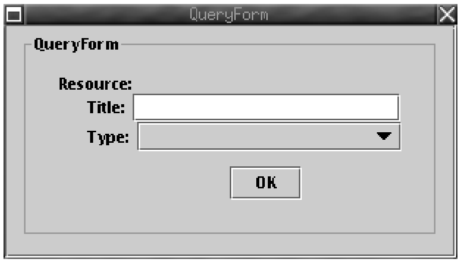


Figure 21: An example RDF form.

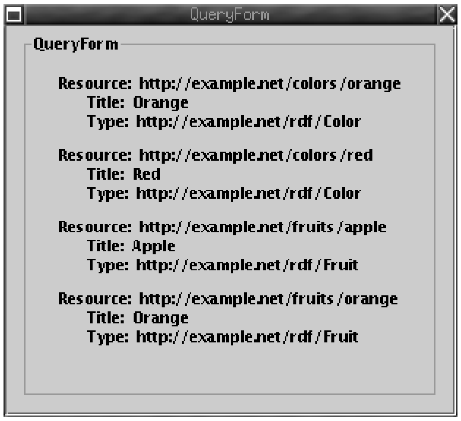


Resource:

Title:

Type:

Figure 22: Query form rendered from an RDF form in Conzilla.

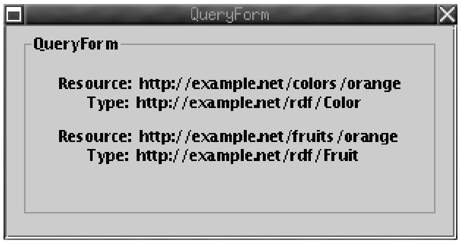


```
Resource: http://example.net/colors/orange
    Title: Orange
    Type: http://example.net/rdf/Color

Resource: http://example.net/colors/red
    Title: Red
    Type: http://example.net/rdf/Color

Resource: http://example.net/fruits/apple
    Title: Apple
    Type: http://example.net/rdf/Fruit

Resource: http://example.net/fruits/orange
    Title: Orange
    Type: http://example.net/rdf/Fruit
```

Figure 23: Result form generated after executing the unaltered query.

Since all four resources in the Edutella provider's RDF knowledge base have type and title information, all of them are returned as the result of the query. For each of the four results, the values bound to the three variables of the query are displayed.

If we now refine our search by entering "Orange" as the title in the query form, we get the result shown in *Figure 24*.

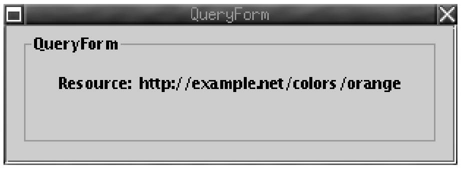


```
Resource: http://example.net/colors/orange
    Type: http://example.net/rdf/Color

Resource: http://example.net/fruits/orange
    Type: http://example.net/rdf/Fruit
```

Figure 24: Result form for the title "Orange".

Here, the literal "Orange" has been substituted for the title variable in the query before execution. The values bound to the remaining two variables are shown as the result.

There are two resources that have the title "Orange", one of which has the type `ex:Color` and the other has the type `ex:Fruit`. If we're only interested in the color orange, we can choose the `ex:Color` as the type in the query form in addition to entering the title "Orange". The result after query execution is shown in *Figure 25*.



```
Resource: http://example.net/colors/orange
```

Figure 25: Result form for the title "Orange" and the type `ex:Color`.

8 Future Perspectives

8.1 Extensions

8.1.1 Form Layout And Style

Nothing has been mentioned so far about the style of forms. As coded in RDF in this work, forms are hierarchical tree-like structures similar to HTML structures. At present the translation of the RDF forms into Java user interfaces is hardcoded.

In analogy with separating out style information of HTML into style sheets (CSS), it would be possible to create separate style information about RDF forms. The benefits of this would be the same as with CSS—flexibility, clarity, and if the style information itself was coded in RDF it would be possible to share, reuse, and recombine style information between forms.

CSS makes use of selectors to specify which style information that should apply to each part of a document [1]. This is a very powerful way to specify style information. By giving general style information a general selector, such as e.g. the `<body>` tag (the actual CSS selector for that would simply be "`body:`"), all children can inherit this information. More specialized children can have style information with a more specific selector, which overrides the general one. In this way a minimum of style information is needed. This is not only good news for lazy style writers but also means that style information can be easily changed and reused.

With the CSS selectors as inspiration, a possible selector system for RDF would be to use class hierarchies, class instances, and individual resources as selectors. The following is an example of each type, respectively:

- All subclasses of `rdf:Seq` should be displayed with a red outline.
- All resources that are instances of `rdf:Seq` should be displayed with a bold font.
- The resource `http://example.net/myresource` should be displayed underlined.

This is not only applicable to RDF forms, but could be applied when visualizing any type of RDF data. When multiple style statements conflict there is a natural hierarchy among these selectors when deciding which style information should take precedence: Class hierarchies are more general than class instances, which in turn are more general than individual resources. In analogy with CSS, the convention would be to let the more specific selectors override the more general ones.

8.1.2 RDF Workflows

The analysis of query and editing processes in this work leads up to the concept of *workflows*. The workflow concept has had an impact on the design of the query management system. Modularity based on the query and edit processing steps ensured flexibility and a possibility to extend the system with a minimum of duplicated effort. But due to time shortage and it being slightly out of scope, the logical extrapolation of the workflow concept was never drawn.

By coding workflows themselves in RDF, the different types of processes would disappear. There would only be workflows, and creating new processes would simply mean to draw a new RDF workflow graph. This would amount to a powerful RDF scripting language.

What's needed before it would be meaningful to code workflows in RDF is an RDF workflow interpretation engine. Something must translate the RDF workflows into processes and execute them.

8.1.3 Editing

Regardless of whether RDF workflows are implemented or not, it would be very useful to extend the query management system with editing capabilities.

The groundwork for this has already been done. The code for mapping the result of a query to variables of a new query has already been written and is part of the query management class library. This is the necessary "glue" between finding a resource to edit and creating a new query designed to find all editable metadata about it.

The RDF form specification needs to be extended in order to handle the additional complications of editing, such as e.g. restrictions on the type of editing that's permitted on a given resource.

As mentioned before, this work is intended to be a part of the Standardized Hyper Adaptable Metadata Editor (SHAME) project, which aims to create a metadata editing and presentation framework for RDF metadata. An early prototype editor has already been created in this project [23].

8.2 Limitations Of The Edutella Query Language

The query management system doesn't require the Edutella system, but it's rather Edutella biased. Although it would be possible to implement the capability to send queries to other metadata systems without changing any of the existing interfaces, queries sent to the Edutella network will likely be the most important ones for any foreseeable future. Edutella is an international open source effort with a very powerful concept.

That being said, Edutella isn't perfect and, being under development, doesn't claim to be. The most serious problem occurs when trying to express queries of the type, "give me the resource `R` that satisfies some condition `C`, and if available give me the metadata `M` about it". The problem is to express, with Edutella's Prolog like query language, that we want the resource `R` even if the metadata `M` isn't found. Normally when we want a resource with some metadata, we express it as "give me the resource `R` that has metadata `M`". To express the first query we have to use an "or" expression, "give me the resource `R` *or* give me the resource `R` that has metadata `M`".

This becomes extremely inefficient when we want a lot of metadata. The number of disjunctive "or" clauses grows exponentially with the number of metadata properties we want about a resource.

It's not possible to express the query efficiently in Prolog. What's needed is the "outer join" expression of database languages like SQL. A reasonable solution would be to extend the Edutella query language with something equivalent.

8.3 Edutella Queries And RDFS

In *Section 5.2.6* it was noted that the class membership of sequence items could not be specified due to the lack of typed containers in RDFS. Although extending RDFS is slightly off topic when discussing query management, there is one interesting aspect of the RDFS limitations.

The purpose of specifying a vocabulary with RDFS is to define valid RDF constructs. In order to check whether a specific instance of e.g. an RDF form satisfies the vocabulary specification of RDF forms, one has to compare the instance with the specification somehow. If this is to be automated, one solution would be to design a query that matches RDF forms. One could then execute the query on the RDF database containing the RDF form instance and see if a form with the specified root resource was found.

The query designed to find RDF forms would then be a kind of vocabulary specification of RDF forms. Edutella queries could be used instead of, or as a complement to, RDFS. In addition to higher expressiveness, Edutella queries provide the possibility of automating the verification of RDF vocabularies.

8.4 Non-Authoritarian Metadata View

8.4.1 External Annotation

By its very nature, RDF metadata provides the ability to express information about other resources. This obviously results in the possibility of annotating read-only information. Everyone is free to say anything about anything.

This has important implications for the freedom of expression on the RDF-based Semantic Web. When carrying out a search, the combined RDF metadata of the whole accessible web will be the database where the information is collected from.

8.4.2 Metadata Authentication

While freedom of expression is very desirable, there must be some way of weighing information when e.g. different RDF sources contradict each other. The only one capable of this is the reader of the information, but in order to form an opinion it is necessary to know from where the information came.

What's needed is some form of system for authenticating the origin of metadata so the receiver can estimate its credibility. This would have to include some form of digital signature that is reasonably difficult to forge.

Although an authentication system like this lies outside the scope of a query management system, it would have implications for the management of queries. There would have to be a way of specifying sources you trust and distrust, and perhaps a way of weighing results from different sources.

9 References

References

[1] Cascading Style Sheets, http://www.w3.org/Style/CSS/

[2] The Apache Ant project, http://ant.apache.org/

[3] The concept browser Conzilla, http://www.conzilla.org/

[4] Nilsson, M., Palmér, M. 1999, Conzilla - Towards a concept browser , CID-53, TRITA-NA-D9911, Department of Numerical Analysis and Computer Science, KTH, Stockholm., http://kmr.nada.kth.se/papers/ConceptualBrowsing/cid_53.pdf

[5] Naeve, A. 2001, The Concept Browser - a new form of Knowledge Management Tool, Proceedings of the 2nd European Web-based Learning Environments Conference (WBLE 2001), Lund, Sweden., http://kmr.nada.kth.se/papers/ConceptualBrowsing/ConceptBrowser.pdf

[6] The Dublin Core Metadata Initiative, http://www.dublincore.org/

[7] The Edutella project, http://edutella.jxta.org/

[8] Nejdl et al. 2002, EDUTELLA: A P2P Networking Infrastructure Based On RDF, WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA., http://edutella.jxta.org/reports/edutella-whitepaper.pdf

[9] The GNU General Public License, http://www.fsf.org/licenses/

[10] Some early ideas for HTML, http://www.w3.org/MarkUp/historical

[11] HTML, http://www.w3.org/MarkUp/

[12] HTML, first version, http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html

[13] HTML, version 2, http://www.rfc-editor.org/rfc/rfc1866.txt

[14] HTML, version 3, http://www.w3.org/TR/REC-html32

[15] HTML, version 4, http://www.w3.org/TR/html401/

[16] The Java language, http://java.sun.com/

[17] The Jena Semantic Web Toolkit, http://www.hpl.hp.com/semweb/jena.htm

[18] Platform for Internet Content Selection (PICS), http://www.w3.org/PICS/

[19] Resource Description Framework (RDF), http://www.w3.org/RDF/

[20] Resource Description Framework (RDF) Model and Syntax, WD-rdf-syntax-971002, http://www.w3.org/TR/WD-rdf-syntax-971002

[21] RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/

[22] The Semantic Web, http://www.w3.org/2001/sw/

[23] Standardized Hyper Adaptable Metadata Editor (SHAME), http://sourceforge.net/projects/shame/

[24] Unified Modeling Language (UML), http://www.omg.org/UML/

[25] Naming and Addressing: URIs, URLs, ..., http://www.w3.org/Addressing/

[26] Uniform Resource Identifiers (URI): Generic Syntax, Request for Comments: 2396, http://www.ietf.org/rfc/rfc2396.txt

[27] Extensible Markup Language (XML), http://www.w3.org/XML/

10 Acknowledgement

I would like to thank Matthias, Ambjörn, and Mikael for all the help, discussions, interesting ideas, and general encouragement.

I would also like to thank everyone (including the wonderful coffee machine) at Uppsala Learning Lab for providing such a nice working environment.

Last but not least, I would of course like to thank my lovely girlfriend Anja for being my lovely girlfriend Anja.