National University *of* Ireland, Galway
Ollscoil na hÉireann, Gaillimh
Department *of* Electronic Engineering
Roinn na hInnealtóireachta Leictreonaí

Ranking Semantic Web Graphs

Author: Aidan Hogan

Supervisor: Dr. Peter Corcoran

B.E. Electronic and Computer Engineering

Final Yr. Thesis

March 05

Declaration of Originality

I hereby declare that this thesis is my original work except where stated

Signature:______________________________________ Date:___________________

Abstract

With an ever-increasing volume of data on the web and more people availing themselves of Internet services, the ability to retrieve data relevant to the user is becoming more and more endangered. There are many researchers working in the area of trying to improve information retrieval within the domain of the Net. Two such areas, coming from different backgrounds and different directions are *Ranking Algorithms* for giving priority in information retrieval services to more content rich pages, and also the *Semantic Web*, a framework within which more structured data can be provided on the Web.

Ranking algorithms on the Web are of great utility to users who require specific information from the Web. Most people requiring such information usually avail themselves of a search engine, the most prevalent being *Google*. Google uses a ranking algorithm called *PageRank* to give priority to more content rich and useful pages in searches, a system which has made the company a household name.

The Semantic Web is a venture in which much research is being pursued. The Semantic Web hopes to bring order to a chaotic Web, by providing a structure to data which is interpretable by computers who can then take some of the workload from users in services such as information retrieval. Endorsed by the *W3C* and a vision of the future of the Web by *Tim Berners-Lee*, work in the Semantic Web is being pursued by many companies. Indeed, Semantic Web technologies are in employment today in a wide range of areas.

The purpose of this project is to try and bring the advantages of structured data and ranking algorithms together to successfully provide an advanced search and browsing tool for different domains of structured data. More specifically, three pre-existing datasets are acquired, indexed, stored and analysed and then a search engine application is applied to each, which provides the user with a ranked results set.

Acknowledgements

I would like to acknowledge the role of my supervisor, Dr. Peter Corcoran in this project and for the advice and support he offered for the duration of the project.

I would also like to thank those working in DERI, Ireland for allowing use of their server arrays and resources for the work described herein. I would especially like to mention the support of Andreas Harth and John Breslin, who offered me advice and support and also technical help with using the servers.

Table of Contents

Section **Page**

Chapter 1: Introduction

This thesis describes the work achieved with the aspiration of creating a search engine tool that can be applied to any Semantic Web dataset which has an inherent “simple” graph characteristic, that is to say a graph consisting of one type of node/resource and one type of edge/link. By indexing and storing the dataset through Semantic Web languages and applying graph based ranking techniques to said, it is possible to create a search engine tool which fully exploits the data retrieval advantages intrinsic to both Semantic Web technologies and graph based ranking algorithms. Presented herein are many areas of research and work in both fields that are innovative and practical and culminate in an application with advanced ranked search and retrieval capabilities for three separate datasets.

This Chapter begins by introducing the two main research areas involved in this project and continues by introducing the three datasets which comprised the use-cases of the search engine system. *Chapters 2* through *5* then deal with the background, motive, design and creation of the various components required to implement said system, and *Chapter 6* presents the end product, three separate search engines for each dataset computing returning ranked results page.

1.1-Ranking Algorithms

Ranking algorithms are employed by search and retrieval systems to order the resources returned by the query. Two main categories of ranking algorithm exist, keyword ranking and links based ranking.

Keyword ranking algorithms, such as *TF-IDF*, are solely based on the keywords that a user enters as a search parameter to the system. They generally aim to try and place numerical values of relevance of each document or resource indexed by the system to the term(s) or phrase(s) in the users query. Usually, this is based on the rarity of usage of a particular word, the frequency of its occurrence in a document, entire phrase matches and/or the positioning of the words in the document (words in title boost relevancy). Those documents with higher scores float to the top of the results page issued to the user.

Links based or graph based ranking algorithms can be applied when there exists connections within the resources being indexed. This can occur when resources link, cite, reference or in some other way mention another equivalent resource. Such connections lead to what will be referred to as *explicit graphs* for the duration of this thesis. An example of this is the World Wide Web where web-pages link each other by means of Hyperlinks. Indeed PageRank, the most prevalent links based ranking algorithm, is used on this premise by Google.

In certain cases, connections can also be inferred by resources sharing a common property or value. This inference can be useful where graph based algorithms would ideally be applied to a database of resources that have no explicit connections. For instance, in a database of authors of research papers, authors could be linked by co-authorship of a paper. That is to say such authors are linked by sharing a common property: having authored the same paper. Such graphs existing within a dataset shall be referred to as *implicit graphs*.

Why, you may ask, are links in the dataset between resources so important? How can they help with finding relevancy to a keyword search entered by the user? The simple answer is that they can't. What they do allow however is a form of peer review, where if one resource links to another, it can be interpreted in the generic sense as a *vote of confidence*.

To be more specific about this idea of peer review, lets examine the example of the Web with web-pages interlinking. Spamming etc. aside, the motives behind Page A linking Page B is that the creator of Page A believes that there is content in Page B that is relevant and useful to someone accessing the page. Therefore, the link can be seen as "a vote of confidence" in Page B by Page A. How exactly these votes are tallied is the subject of *Section 2.1*.

1.2-The Semantic Web

The Semantic Web is an attempt to bring order to the ever sprawling Web by, among other things, providing a framework for the online publication of structured data. The contemporary Internet is a vast expanse of HTML web-page. HTML is a mark-up language which means that various components of the document are tagged – enclosed by specific HTML elements. This allows web-browsers to find what part of the document refers to the header, background, body, links etc. This is essentially as far as the structure extends, with probably vast amounts of data left unstructured in prose throughout the body of the page.

XML, or *eXtensible Markup Language* is again a markup language but is not restricted to tags that only a web-browser needs to display a document. XML documents are intended to describe (meta)data as opposed to HTML which uses tags to display data in a desirable way. XML essentially allows data exchange esp. over the Web, and its power lies in the "X", eXtensible. This means that it can be used for describing almost any generic data. For instance in Fig. 1.1, an example XML snippet is used by a (very small) video store to describe their products. It describes three products and some of their pertinent properties. It is quite easily interpreted by a human reader, however using technologies developed by various working groups, technologies such as *XPath*, *XQuery* and *XSLT*, applications can be build to interpret, edit, search, create etc. such documents.

Two important parts of XML are *XML Schema* and *namespaces*. XML schema allow a greater definition of what each tag represents, what properties are applicable to it (if any) and restricts XML documents to that structure. So for instance, an XML Schema could be created to restrict the `rentable` attribute in Fig. 1.1 to exclusively `Boolean` values, etc.

XML namespaces allow elements to be mapped to a unique identifier, a URI. This can be useful to resolve any ambiguity that may exist as to what a tag refers to. It also helps the interoperability and inter-interpretability of XML files, i.e. where files from different locations using the same namespace (scheme of URIs) and tag identifiers etc. can be interpreted by the same applications. This stems from the fact that data can be linked through the Web by using a URI. The same types of property or value are mapped to the same URI.

However, the Semantic Web goes another step, with its main data exchange format called *RDF*, or *Resource Description Framework* [1]. RDF is not a single language, but is instead a *framework* for data publication. It is again intended for when data will need to be processed by software agents and has vast scope for interoperability. This interoperability comes from its strict syntax for representing data, and the advantages of this interoperability are exploited by the work presented in this thesis.

```
 <?xml version="1.0"?>
<products>
      <films>
          <film rentable = “true” format = “dvd”>
                 <title>Speed 2</title>
                 <copies>3</copies>
                 <price>3.00</price>
          </film>
          <film rentable = “false” format = “video”>
                 <title>Spiceworld</title>
                 <copies>1</copies>
                 <price>29.99</price>
          </film>
      </films>
      <games>
            <game rentable = “true” platform = “xbox”>
                 <title>Mario the Hedgehog</title>
                 <price>2.00</price>
                 <copies>2</copies>
            </game>
      </games>
</products>
```

Figure 1.1 An example XML snippet describing the product of a video rental store

RDF centres around the idea of triples, which are a simple construct consisting of subject, predicate and object. It is enforcing such constructs in the structure of data and also selective use of compulsory namespacing, discussed later, which allows RDF its advantage over XML.

Triple elements are at the core of all data exchange. If one thinks of them in the context of the English language, the following slightly simplified English phrase:

Toyota is **owned by Tim**

has subject **Toyota**, predicate **owned by** and object **Tim**. The subject indicates what resource or entity is being described, the predicate offers the type of property of that resource which is being described and the object gives the value of that property. Objects themselves can become subjects in data, i.e.

Tim owns Toyota

Tim authored Hamlet

Tim is gender female

In both lines **Tim** is now the subject, predicates are **owns, authored** and **gender** and objects are **Toyota, Hamlet** and **female**. Sometimes, RDF also allows context to be specified. Context is essentially the source of the data i.e. who authored the data or where it was found. This is the basis of semantics in the Semantic Web.

RDF also makes use of URIs and namespaces. All resources should be given a specific property called **type**. Ideally, all resources should also be given an ID, a URI. If not they should be provided a blank node identifier. Also, all properties must be given their own URI. So now we have

```
<http://www.ppl.com/Tim> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.animals.com/Person> .
<http://www.ppl.com/Tim> <http://www.generic.com/name> “Tim” .
<http://www.ppl.com/Tim>  <http://www.humanproperties.com/is_gender> <http://www.genders.com/female> .
<http://www.ppl.com/Tim> <http://www.verbs.com/authored> “Hamlet” .
<http://www.ppl.com/Tim> <http://www.ownership.com/owns> <http://www.move.com/Toyota> .
<http://www.move.com/Toyota> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.move.com/car> .
<http://www.move.com/Toyota> <http://www.generic.com/name> “Toyota” .
```

Figure 1.2 An example snippet of RDF-NTriples describing a person and his car

This is an example of RDF-NTriples, one of the existing RDF languages which constitutes an important aspect of this project. As can be seen there are three entities to a line, each one part of a triple. The first entity is always the subject and so it follows for predicate and object respectively. All subjects must be unique identifiers, either URIs or blank nodes, properties must be URIs and finally objects can be either URIs, blank-nodes or literals. URIs are enclosed by <> characters, blank nodes are generally an integer prefixed by `_:boid` e.g. `_:boid5`, and finally literals can be any form of text as long as certain characters are properly escaped and are enclosed by inverted commas.

All the data about Tim and his automotive tendencies are still included but now in a computer interpretable form. If Tim uses the same URIs as other people use for his predicates and types (but not his own ID, it refers to a unique entity: himself) and publishes the data on the web then applications which find it, such as mediators or search engines, will be able to index away his data with all the other data using the same URI naming scheme. This is one essential aspect of the Semantic Web, which has its roots in XML. With the growth of various URI naming schemes for different types of data, many human and software agents are beginning the publishing, on the Web, of RDF data.

Again RDF is not a single language. Some of the languages within the framework include RDF-XML, an XML based language and RDF-N3, (of which RDF-NTriples is a subset) which allows single facts of data to be encoded into lines. RDF-XML also plays an important part in this project as a stepping stone from other data-types.

1.3-Datasets Used

In order to bring about this synergy between Semantic Web technologies and ranking algorithms, some datasets were needed to experiment on. These datasets essentially needed to be data about uniform resources using some form of uniform description language. The datasets also needed to feature inherent graph characteristics. That is to say the resources described need to have some property (predicate) whose value (object) was an equivalent resource (a resource with the same value for the property "type").

Three datasets were deemed suitable, *CiteSeer*, *FOAF* data from a site called *GreatestJournal* and also *US patent* data. These datasets were prepared, analysed, indexed and eventually a search and retrieval tool built on top of them.

1.3.1-Citeseer

CiteSeer is an online search engine for published material in the areas of computer science and information retrieval. It offers search and retrieval of metadata about research papers and other published works in these areas. CiteSeer offers the download of a compressed archive of such metadata from their site[1]. This data is in the form OAI which is an XML-like language and features the metadata of nearly 600,000 research papers. There are two graphs inherent in CiteSeer, one derived from research papers citing other papers and the other derived from authors having co-authored with one another. The former graph is an example of an *explicit* graph, the latter an *implicit* graph.

[1] http://citeseer.ist.psu.edu/oai.html

1.3.2-GreatestJournal FOAF

FOAF, or *Friend of a Friend* data is a type of RDF data using URIs from a specified namespace, the FOAF namespace[2]. Such data is usually expressed in RDF-XML and is primarily concerned with publishing data regarding people.

The site *GreatestJournal*[3] allows users to create their own "journal", which is essentially a community accessible web-page. It allows the editing of other users journals if not restricted and encourages growth of communities. The site, at time of writing, boasts about 1.3 million users. GreatestJournal dynamically creates a FOAF file for each new user it registers. The FOAF file contains data such as username, name, interests and most importantly for our purposes, a `foaf:knows` property, the object of which is an element referring to another user of GreatestJournal which the user has specified as a "friend". This property creates the graph which is of interest to us.

This data is not available as a single download, but instead each FOAF file is provided at an address accessible over the Web. This data can be retrieved by crawling as a user's FOAF file feature links to the FOAF files of users it has indicated as friends. However crawling will not return the full dataset, as some introverted users may not be linked to by another user.

1.3.3-US Patents

The *USPTO*, *US Patents office*[4] indexes and offers search and retrieval tools for various metadata about patents. USPTO has a data store of patent data for millions of patents, ranging between four different formats (that is to say the format used has changed four times, and some patent data is available in one, some only available in another, etc.). One of these formats is XML based and has been in use since April 2005[5]. The data is available through an FTP server[6] as compressed archives of a weeks patent data. Patents often cite other patents as relevant, creating a graph of patents.

[2] http://xmlns.com/foaf/0.1/
[3] http://www.greatestjournal.com/
[4] http://www.uspto.gov/
[5] http://www.uspto.gov/web/menu/patdata.html
[6] ftp://ftp.uspto.gov/pub/patdata/

Chapter 2: The Ranking Component

2.1-PageRank

PageRank is a links based ranking algorithm employed by Google to rank and return results to a user from its indexed web-pages, as mentioned in *Section 1.1* [2]. Such a system for our search and retrieval aspirations would be pivotal, allowing a return of ranked result resources from our three datasets. The resources (papers, authors, users and patents) become nodes in a graph linked by different edges (cites, co-authors, knows and references). These graphs, which can be described in RDF and can be extracted from their source datasets (as will be described in *Chapter 4*), can also be analysed by a technique based on PageRank.

2.1.1-Rationale behind PageRank

PageRank uses links between nodes as a means of determining the *prevalence* or *importance* of a node. A node that has many links to it is obviously quite relevant to many other nodes. This usually means that the node is perhaps content rich, has great utility or is somehow relevant to its peer nodes. It uses such analyses to place a numeric value called a PageRank score, on each node's prominence.

Of course with all this talk of nodes, graphs, RDF, predicates and later talk of eigenvectors, connectivity matrices and quadratic extrapolation methods, it occludes appreciation of the human aspect of all links based ranking algorithms. All the resources (nodes) and all the links (edges) have a human source. Some person authored the paper that's being ranked and cited the papers that were relevant, someone was involved with the referencing of one patent by another, someone was involved in the decision to co-author with another, and someone decided that another's journal was of interest to them. All things considered, algorithms like PageRank, based in the world of computers, logic and math, exploit these somewhat subjective human decisions to provide scores for resources that humans find relevant. It is essentially the correlation of a peer review or a system of voting, an idea that helps rationalise this talk of computers deciding relevancy of results to human users.

The basic premise of PageRank is that nodes begin with equal importance, an equal PageRank score. The importance of a node is then calculated solely on its inlinks and their source. Intuitively, the outlinks of a node cannot affect its importance (what papers a paper cites does not make much difference, as such, to its importance; the paper would be important however if many other important papers cite it). Nodes are also banned from voting for themselves. PageRank is not simply a case of counting inlinks however, again the **source** of the inlinks are of vital importance in calculation. An inlink from a relatively important node can be just as valuable as many inlinks from relatively unimportant nodes. To clarify with an example, if a research paper is deemed as an extremely important paper, the papers it cites should be regarded as also important. On the other hand, the citations of less important papers demand less respect. Taking this into account the *voting power* of a node is basically its PageRank score. More important nodes have more say.

Also of importance is the number of outlinks the inlinking node makes. For instance, if node A links to 100 other nodes, node A's vote will be evenly divided over all the nodes. So if you have been linked by node A, even if node A is of high importance, you will receive only 100th of node A's vote.

2.1.2-Process of Calculation

PageRank is an iterative computation which begins with all nodes being equal. In each successive iteration, the score of node A is determined as a summation of the PageRank score in the previous iteration of all the nodes that link to node A divided by their number of outlinks.

There is one complication, which is a damping factor, usually signified by *d*. It has a numeric value usually agreed upon as 0.85. A node's voting power is in fact only 0.85 of its PageRank score. The other 0.15 is split evenly amongst all other nodes. There are now two types of links in the graph, *strong* and *weak*. Strong links are the links derived from the dataset. Weak links are artificially created by the damping factor and link all nodes to all other nodes (and itself). This is an important aspect in the calculations which are described herein.

Taking an example, the code in *Fig. 2.1* is a fictitious snippet of an RDF-NTriple graph description derived from the CiteSeer dataset. The method of derivation is described later in *Section 3.1*. This graph can be interpreted as paper with identifier `oai:CiteSeerPSU:1` cites paper with identifier `oai:CiteSeerPSU:2`, etc. From now on `oai:CiteSeerPSU:1` shall be referred to as node 1 or paper 1, and so on. This graph can be visualised as illustrated in *Fig. 2.2*, which incorporates the node index of *Table 2.1*.

```
<oai:CiteSeerPSU:1> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:2> .
<oai:CiteSeerPSU:1> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:3> .
<oai:CiteSeerPSU:2> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:1> .
<oai:CiteSeerPSU:2> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:4> .
<oai:CiteSeerPSU:3> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:4> .
```

Figure 2.1 Source NTriples Graph

The first task involves parsing the NTriples data to get all the pertinent graph details. This is described in *Section 2.2*. In doing so, the aim is to create both a node index, which maps an integer value onto the node's identifier (*Table 2.1*), and a connectivity matrix (Initial Matrix in *Table 2.2*) which defines which nodes link which. For the duration of the determination of the final PageRank scores, the node is known by a number and not a string identifier.

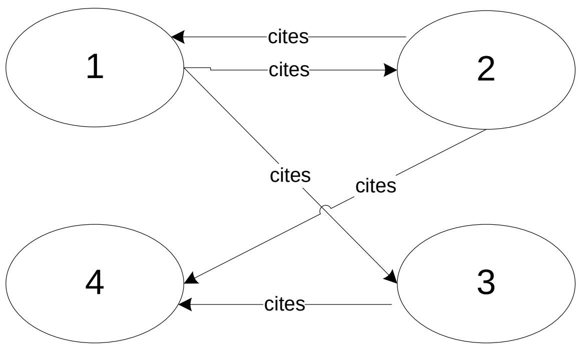


Figure 2.2 Node Index

Index	Node Identifier
1	oai:CiteseerPSU:1
2	oai:CiteseerPSU:2
3	oai:CiteseerPSU:3
4	oai:CiteseerPSU:4

Table 2.1. Node Index

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 \end{array}$$

Initial Matrix

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0.5 & 0.5 & 0 \\ 2 & 0.5 & 0 & 0 & 0.5 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 \end{array}$$

Normalised Matrix

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0.0375 & 0.4625 & 0.4625 & 0.0375 \\ 2 & 0.4625 & 0.0375 & 0.0375 & 0.4625 \\ 3 & 0.0375 & 0.0375 & 0.0375 & 0.8875 \\ 4 & 0.25 & 0.25 & 0.25 & 0.25 \end{array}$$

Matrix with Damping Factor

Table 2.2. Connectivity Matrices

Configuring the connectivity matrix is the next step. From parsing the source data, the initial connectivity matrix is derived. Seen in *Table 2.2*, the initial matrix is simply a *(n x n)* matrix of 1's and 0's where *n* is the number of nodes being ranked, 1 indicates a link and 0 indicates an absence thereof. Index *(i,j)* of the matrix indicates that node *i* links node *j* (node *i* and node *j* referring to the nodes of the same value in the node index, *Table 2.1*).

Each row must then be normalized to a sum of 1. The value in index *(i,j)* now refers to the proportion of node *i*'s vote that will be given to node *j* disregarding the damping factor. Each link in the row is treated equally.

Finally for configuring the connectivity matrix, the damping factor *d* must be factored in. For every row, each non-zero entry is multiplied by the damping factor *(0.85)* [2]. Every entry in the row is then evenly distributed the weak link *(0.15)*. That is to say, each entry in the row, zero or non-zero, has *0.15/n* added. This results in the third matrix displayed in *Table 2.2*. The value of index *(i,j)* is the final version of the proportion of node *i*'s vote that will be given to node *j*. Of interest is the fact that all entries are non-zero, and that each row adds to 1.

One may note that row 4 does not follow the above rules. Node 4 is in fact a node with no outlinks. Referring to *Fig. 2.2* should confirm this. In this case, node 4's vote is evenly distributed. In fact, node 4 has no impact on the ranking and some parties promote removing such nodes from the proceedings, however I found it more convenient for my own purposes to leave such nodes in and split their vote evenly.

After this step follow the iterative calculations. The PageRank score is an *n* length vector, one score for each node. Initially the score for each node is initialized to *1/n*. The value for index *i* of the vector is of course, the score of node *i*. The following equation applies;

$$PR_{k+1}(i) = \sum_{j \in IN_i} (PR_k(j) * CM(j,i)) \qquad (2.1)$$

where *i* is a node, IN_i is the set of inlinks to *i, PR* is the PageRank score vector, *k* is the number of iterations and *CM* is the connectivity matrix with the damping factor included.

Following these calculations is equivalent to calculating the *first eigenvector* of the third connectivity matrix in *Table 2.2*, using the Power Method. This is no coincidence as a mathematical interpretation of the PageRank score vector, is indeed the said eigenvector, and said connectivity matrix can be viewed as a *Markov chain* which is made *ergodic* by the damping factor, a Markov chain representing next state probabilities for a random walk from node to node through the links in the graph. That is to say that the element in index *(i,j)* in the term of a random walk is in fact the probability of landing yourself at node *j* given that you are currently situated at node *i.* The damping factor can be interpreted as the probability that a random surfer will reset the walk i.e. stop following the "strong" links and teleport to a random node. With this in mind the first eigenvector of the Markov chain equates to the long term probabilities of a random walker being at the given node [2].

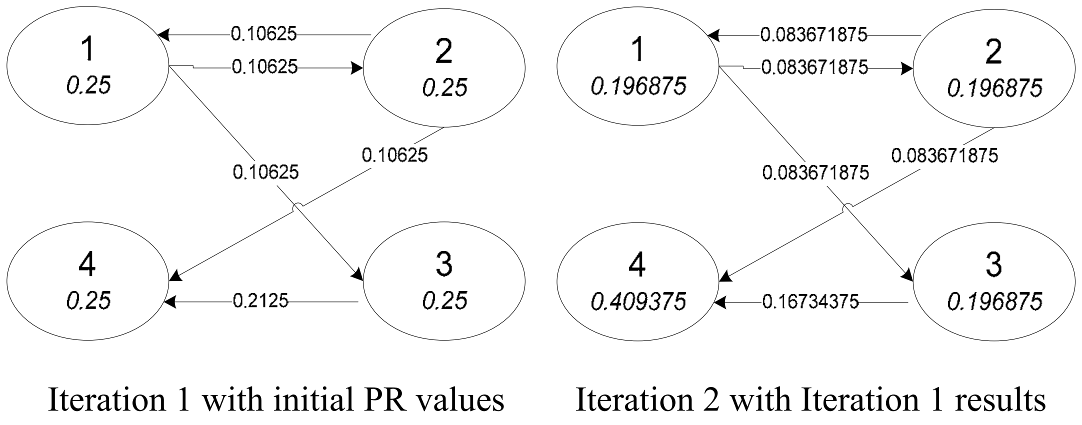


Figure 2.3 First two iterations

Fig. 2.3 illustrates the propagation of PageRank scores through strong links over the first two iterations. Included within the oval, below the node index, is the value of PageRank from the previous iteration (PR_{k-i}). Before iteration 1, all PageRank scores are evenly set to *0.25* each. The links now indicate the numerical value propagated through that link, each node *i* splitting its voting power (which has a value of $d * PR_{k-i}(i)$ in iteration *k*) over all its outlinks.

The PageRank of the node in any iteration is the sum of all the incoming propagated score, plus a minimum value arisen by the damping factor *d* and nodes with no inlinks splitting their vote over all nodes. This has the value of *0.090625* in iteration 1 and was left out of the figure for clarity.

The PageRank values will settle at a fixpoint of Node 1 = 0.2, Node 2 = 0.2, Node 3 = 0.2

and Node 4 = 0.4, and so node 4 is deemed the most important, which is intuitive. Also intuitive in this deliberately simple outcome is the results. The value of node 2 must equal node 3 as they have an equal input, one link from node 1. Also, node 1 must be equal as it receives half the input of node 2, which is the same as the input to node 2 and three. A brief examination of *Fig. 2.3* can confirm that nodes 1, 2 and 3 have and always will have the same value input, and that the ratios of the inputs will not change.

It follows that node 4 must have a fixpoint value of twice the other nodes as it has an input of all of node 3, and an input the same as node 1. One could be forgiven initially for thinking that the iterations would never converge to a fixpoint but one must remember that even though node 4 would seem to continue its growth, draining the score from the other nodes through node 3, it continuously spreads its score evenly over all the other nodes thanks to the exception, mentioned earlier, for nodes with no outlinks. Once this strikes a balance (aided by the damping factor) the calculations have reached their fixpoint.

The advantage of using the damping factor is shown empirically in *Fig. 2.4*, a replica of *Fig. 2.3* without use of the damping factor. As can be seen after one iteration, the PR values are much further from the fixpoint than in *Fig. 2.3*. Remember again, the PR values are boosted by the PR value of node 4 being evenly distributed, which is not shown for brevity. This can be explained mathematically by the damping factor being responsible for setting an upper limit of the second eigenvector by a factor of *(1-d).* The increased distance between the first and subsequent eigenvectors allows fast convergence.

The damping factor also helps when subgraphs exist. Such subgraphs are linked from the main graph, but do not link back to it. An example is illustrated in *Fig. 2.5* where nodes 4 and 5 form a subgraph. The damping factor forces the redistribution of PR score through the "weak" links it creates. Without it, all the score would drain into it from nodes 2 and 3.

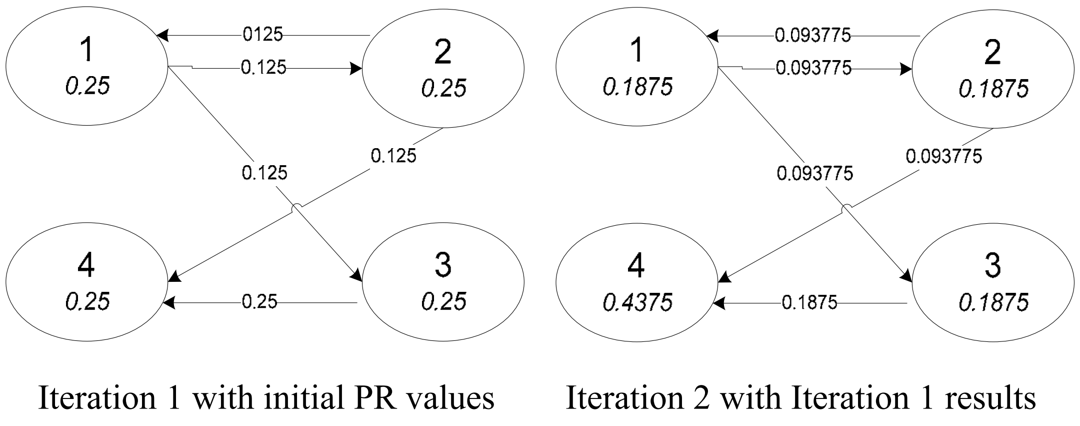


Figure 2.4 First two iterations without use of damping factor

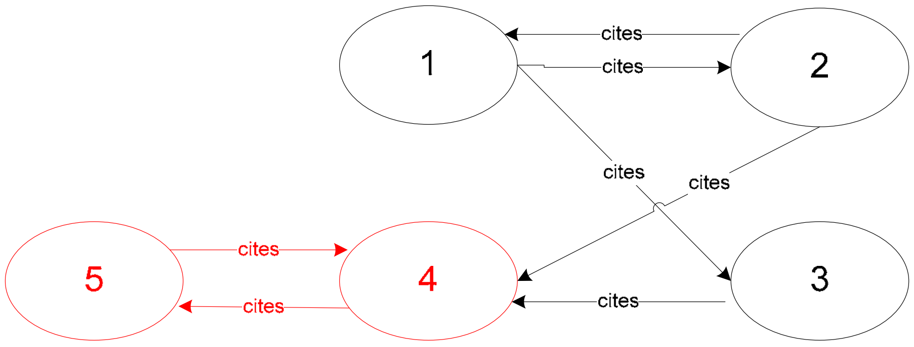


Figure 2.5 Leeching subgraph

Finally the PageRank scores are mapped back to the string identifier for each node, ready to be interpreted by a front-end application.

2.1.3-Local Ranking

Local ranking, or *topic-sensitive* ranking allows more fine tuned results for a user searching for resources on a particular topic [3]. Local ranking is essentially the same process we have discussed (which may be referred to as *global* ranking) except that it is carried out on a subset of the graph.

Lets propose that a user wishes to search for papers relating to the "Semantic Web" and that a keyword query to that effect is entered. A system of global ranking will get the identifiers of all the papers with keyword hits "Semantic Web" and check them off against a pre-computed global ranking score table. The effect can be the possible return of a result set with papers at the top that are deemed very important independently of the topic but only in fleeting mention the “Semantic Web”. For example, a paper could be a very important paper on the topic of databases, be cited by lots of other important database papers and give a cursory mention of the phrase “Semantic Web”. Such a paper could not to be said to be entirely relevant to the topic the user requires results on.

This can be quite a common occurrence where results returned are indeed important papers, and do feature the keywords required but are not in fact relevant to the area the user requires searching. Local ranking solves this issue.

Instead of a pre-computed lookup table of global ranking results, local ranking allows dynamic ranking of topical subgraphs. Returning to our example, if the user enters a query for papers in the area of the “Semantic Web”, all the papers with a keyword hit are isolated, and only links between those papers are used in ranking, creating an example of a topical subgraph. The said important paper on databases will be in the subgraph, but will not be benefited by its links from database papers. Instead, only those papers heavily linked by other semantic web papers, and so relevant to the topic, will be highly scored.

This is illustrated in *Fig. 2.6* which shows a subgraph of nodes 1-10 referring to research papers which have caused a keyword query match and which have been isolated from the main graph. Nodes 5 and 6 have also caused a keyword match. Although these nodes shown in blue are heavily cited in the main graph, these external citations (red) are not counted in local ranking and so are completely irrelevant. All that matters are links between the 10 nodes, links in black. Thus resources referring to nodes such as 4, 8 and 9 will be returned at the top of the results page, with resources referring to nodes 5 and 6 featuring near the bottom.

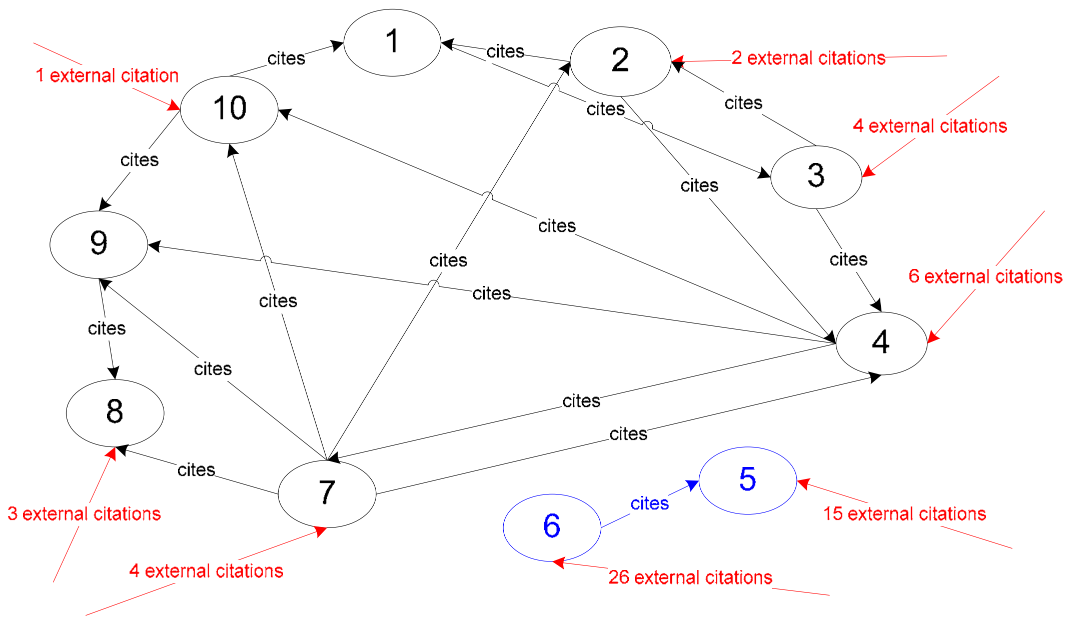


Figure 2.6 Isolation of a subgraph punishing nodes 5 and 6

```
 Node : node8 rank : 0.1482085321911664
Node : node4 rank : 0.14147615827245175
Node : node3 rank : 0.12712997218772998
Node : node9 rank : 0.12133470236191726
 Node : node1 rank : 0.1112559080227273
Node : node2 rank : 0.09911451197995516
Node : node10 rank : 0.08523494510313026
Node : node7 rank : 0.07290457520894209
 Node : node5 rank : 0.0606389092522675
Node : node6 rank : 0.03279450447144371
```

Figure 2.7 Results of subgraph punishing nodes 5 and 6

2.1.4-Clustering

An interesting experiment carried out for this project included utilising the ranking score propagation characteristics to cluster papers into topics which they would not be associated with by keyword alone.

Lets propose again that there was an important paper, but this time was in the area of the semantic web, but in the metadata associated with the paper in the CiteSeer archive, said keywords did not feature (the converse of the problem solved by local ranking). A user enters a query for papers in the area, and the aforementioned paper does not appear.

Clustering aims to help this paper get the placement on the results page it deserves. Clustering involves again taking the topical subgraph, but including those nodes a certain low number of hops away. These hops can also be taken in the inlink or outlink direction.

Fig.2.7 helps illustrate this. It depicts a scenario where nodes 1-9 are returned as keyword matches. No content with just these nine, clustering attempts to add more surrounding nodes into the subgraph. In *Fig. 2.7*, the black links are used in ranking but do not affect subclustering. Node 10 is an example of an important paper with regards the topic of the query, but not featuring the keywords of the query. Nodes 11 – 15 help illustrate the effects of the number of hops taken, and the direction of clustering. All red links could be used by inlink clustering, which will add to the subgraph the node (also in red) from hence the red link originated. All blue links are used by outlink clustering to add the node (again in blue) that they point to. Node 10 is green as it could be added by either inlink or outlink clustering. Black links are not used for clustering. Nodes 11 and 12 would be added by 2 hop inlink clustering. Node 13 would require 3 hop inlink clustering. Node 15 would require 1 hop outlink clustering, whilst finally node 14 would require 2 hop outlink clustering. The link shown between node 11 and 12 would not be used in clustering, as both nodes would be added at the same time by inlink clustering. Once a node is added, all of its links to other nodes in the subgraph are available for ranking.

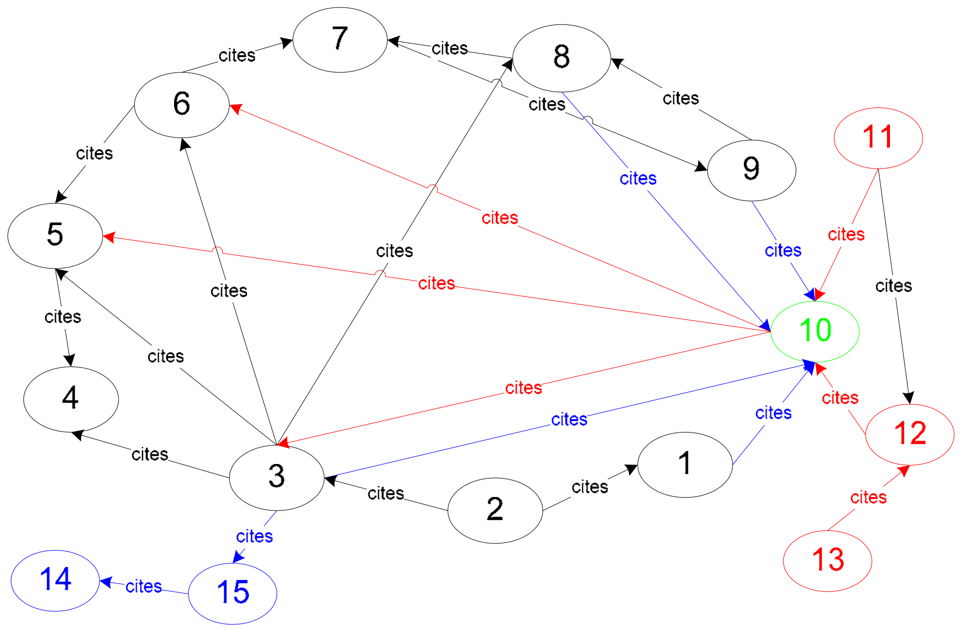


Figure 2.7 Extension of subgraph into other related nodes

As can be seen from the nature of the graph in *Fig 2.7*, one hop clustering in either or both directions will add node 10, a node of high relevance to the topic but not a keyword query hit. The purpose of further hops would be to prevent the restriction of such a node by not having all of its inlinks included. Nodes 1-9 would have all of their remaining inlinks from the external graph added in the clustering step only that added node 10.

It should be noted however that *Fig 2.7* is only an example to illustrate the procedure and purpose of the technique. In real graph terms, if one were to go more than 2 hops away, off-topic results would most certainly be returned quite high on the list. If many more hops are made, the entire graph would be returned in the form of a global rank.

2.2-NTriples Graph Ranking Package

To apply the techniques explained in *Section 2.1* to the applications we desire, a Java package was needed to implement PageRank type algorithms. The processes involved in said application are illustrated in *Fig 2.8*.

The first step in the application would be to create an NTriples parser to allow the input of an NTriples graph file to the application. This was a trivial exercise as the nature of NTriples dictates that resources must be enclosed by < > characters. Utilising the `java.util.StringTokenizer` class with these characters for each line read in from file I/O, isolating the identifiers of the resources and those they link to from the file becomes a straightforward exercise. Indeed, the interoperability of RDF data begins to shine. The same parsing methods work for any NTriples file describing a graph.

Unfortunately however, in developing the procedures that follow said parsing, some major obstacles arose, most stemming from the sheer physical size of the graphs that such a package would be attempting to rank. For instance, the CiteSeer graph of papers features about 570,000 nodes and well over a million links. The NTriples file which describes this graph is about 127MB. Early attempts for the purposes of this project to rank such graphs attempted to closely follow the above described techniques.

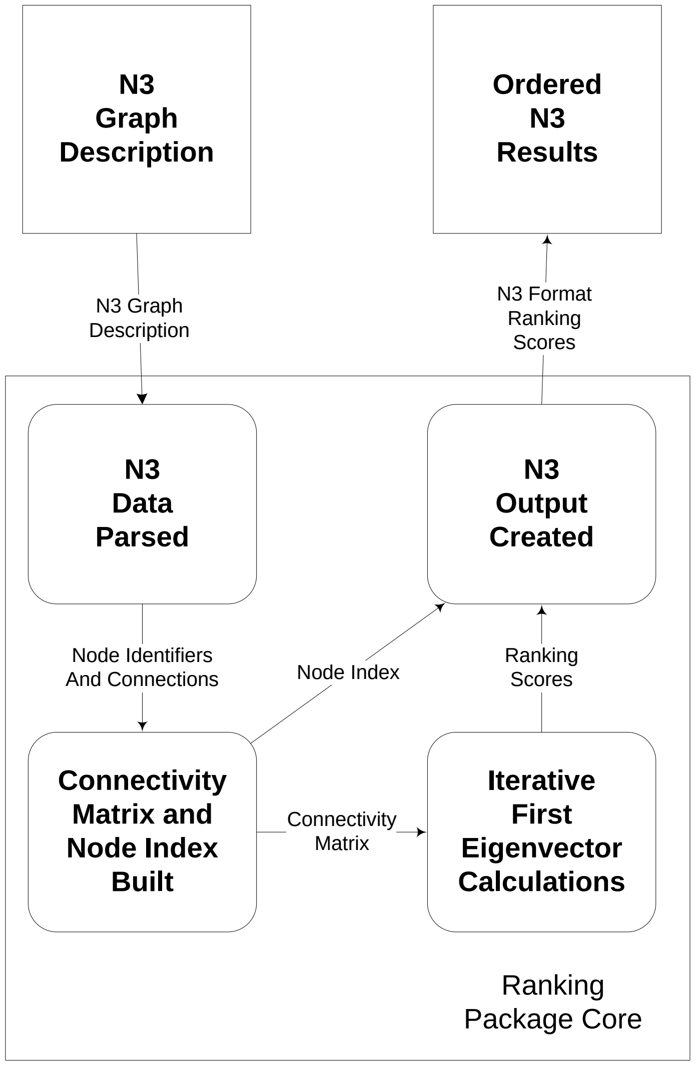


Figure 2.8 Process of ranking NTriples graphs

As has been mentioned in *Section 2.1.2*, the PageRank score vector is the first eigenvector of the connectivity matrix with the damping factor included. Early attempts to calculate said eigenvector were aiming to use an available Java matrix utility package called JAMA[7], however this package failed at finding the first eigenvector of matrices in the order of only 1000 x 1000. Clearly the package was not a viable solution.

Looking to optimise the process of ranking, a brand new package was created from scratch. This package had to be optimised to be memory efficient and computationally efficient. Calculating the eigenvectors of a matrix can be computationally expensive and representing

[7] http://math.nist.gov/javanumerics/jama/

connectivity matrices can be overbearing on memory resources. For instance a traditional square matrix of even a primitive data type in Java for a graph of 500,000 nodes would require 2.5 x 10^{10} or 250 trillion entries. To create this, a heap space of somewhere in the order of 8 terabytes of heap-space would have to be available to the application to load the matrix. Before computational concerns could be addressed, the memory issues had to be addressed lest the algorithm couldn't load any data for computation!

2.2.1-Memory Optimisation

A very simple work around was implemented for loading in the connectivity matrix. Instead of a square matrix, two flat matrices are used. One flat matrix holds inlinks, and its inverse flat matrix holds outlinks. In row *i* of the inlink flat matrix is a list of all the indexes of all the nodes which *i* is linked by and conversely in row *i* of the outlink flat matrix is a list of all the nodes which *i* links. *Table 2.3* illustrates the migration from a traditional square connectivity matrix to two flat matrices.

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ 2 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \qquad \begin{array}{c|cc|} 1 & \mathbf{2} & \mathbf{3} \\ 2 & \mathbf{1} & \mathbf{4} \\ 3 & \mathbf{4} & \\ 4 & & \end{array} \qquad \begin{array}{c|cc|} 1 & \mathbf{2} & \\ 2 & \mathbf{1} & \\ 3 & \mathbf{1} & \\ 4 & \mathbf{2} & \mathbf{3} \end{array}$$

Traditional Matrix **Outlinks Flat Matrix** **Inlinks Flat Matrix**

Table 2.3. Equivalent square matrix and dual flat matrix representations

Although the same data is repeated in the two flat matrices, both are necessary to ease computational requirements. In the square matrix, lookups of the inlinks of a node require looking down a column. Performing the same operation without the second inlinks matrix, the algorithm would have to check every entry in the outlinks flat matrix for the index of the node. The same is true for outlinks. Together however, both matrices offer even faster lookups for links in either direction for a particular node than the connectivity matrix can.

The real utility of the dual flat matrix connectivity format is the savings in memory. Thanks to the extremely sparse nature of the square matrix, vast improvements are made. Each flat matrix has the same number of entries as there are links within the graph, so for a dataset like the CiteSeer dataset, with over 300,000 nodes and about 1.3 million links the total number of entries in both matrices is 2.6 million. This is 100 million times smaller than the square matrix.

The total memory requirements for applying such analysis to a graph of *n* nodes and *m* links is a `java.util.Vector` of *n* node identifiers for lookups of the identifier of a node by its index, `java.util.Hashtable` of *n* buckets for lookups of the index of a node by its identifier, two *n* length array of doubles for the current and previous ranking score vector and two flat matrices with *m* entries each, represented as an array of `Vectors`. An additional requirement which is justified later in *Section 2.2.2.1* is for an *n x 3* array for storing the answers of old iterations used by a convergence acceleration method called Quadratic Extrapolation[4]. This method also required the allocation of 6 *n* length arrays for internal computations. Whilst this was a major step forward from 8 terabytes, heap space of approximately 0.5GB was still required. Thankfully, there were servers available for use by the algorithm with RAM of 2 GB.

2.2.2-Computational Optimisations

With a new memory efficient means of loading into memory the graph's structure, computational concerns were paramount. The flat matrix' outlinks cannot be normalised, nor can a damping factor be included as was illustrated in *Table 2.2*. To do so would require a square matrix, which has already been deemed unfeasible. To analyse the data now represented in this memory efficient manner required a whole new rethink on the process of ranking.

Data readily available to the algorithm included each nodes inlinks and outlinks, and also the total number of outlinks of each node. In each iteration, the damping factor and universal even links from nodes with no outlinks would have to be manually factored into the equation. At first, in each iteration, for each node *i*, a complete search of all of rows of the outlinks flat matrix was completed.

If the outlinks in row *j* did not include *i,* the new ranking score for *i* in iteration *k* would be added with the following value;

$$(1-d)/n * R_{k-1}(j) \qquad (2.2)$$

where *n* is the total number of nodes in the graph and $R_{k-1}(j)$ is the ranking score of node *j* from the previous iteration.

If the outlinks in row j did include i, the ranking score for i would be added with

$$(1-d)/n * R_{k-1}(j) + d/o_j * R_{k-1}(j) \qquad (2.3)$$

where the same notation applies again and o_j is the number of outlinks node j has.

The third case was where row j was empty and node j had no outlinks. In this case the value added was

$$R_{k-1}(j) / n \qquad (2.4)$$

allowing even distribution of node j's vote.

The convergence of the ranking vector was measured by a value called the *L1 residual* [2]. This is a very simple mathematical property which takes the summation of the absolute differences, of all the elements in the ranking vector, between the freshly completed iteration and the previous iteration. This is normalised so that the first L1 residual value is equal to one. When this value reaches a certain tolerance, usually 0.01, the calculations end.

Whilst all the above is a sound means of computing the ranking score vector, it proved to be very computationally intensive. The analysis of the CiteSeer data took about 60 hours of solid computation to derive a result. This did not bode well for the dynamic ranking aspirations (runtime local ranking and clustering) of this project. In order to attempt to accelerate convergence, a method called *Quadratic Extrapolation* was employed.

2.2.2.1-Quadratic Extrapolation

Quadratic Extrapolation is a method by which estimations are made of the non-principal eigenvectors (second and third) and are subtracted from the current iteration estimation. This can lead to a saving of computation time in the order of 30% [4].

The method takes the most recently calculated ranking vector, and the three previously computed ones (answers from iterations k down to $k-3$). A $2 \times n$ matrix Y is then filled with the difference between the ranking vectors from iterations $k-3$ and $k-2$ and the differences between answer vectors from iterations $k-3$ and $k-1$. A QR decomposition of Y using the Gram-Schmidt method is then employed. The resulting Q matrix is then used to compute a 2

x 1 vector *γ* by pre-multiplying a vector with the differences between ranking vectors from iterations *k* and *k-2* by *–Q* transposed. The two values in *γ* are combined with a third value γ_3 which equals 1 to give three values for *β* as shown in *Equations 2.5, 2.6* and *2.7*. *Equation 2.8* shows the final derivation of the eigenvector estimation, where *R* is the ranking score vector (first eigenvector). The *β* values act as ratios of which previous iteration estimates are of the actual principal eigenvector.

$$\beta_0 = \gamma_3 + \gamma_2 + \gamma_1 \quad (2.5)$$

$$\beta_1 = \gamma_3 + \gamma_2 \quad (2.6)$$

$$\beta_2 = \gamma_3 \quad (2.7)$$

$$R_{k+1} = \quad R_{k-2} * \beta_0 \; + \; R_{k-1} * \beta_1 \; + \; R_k * \beta_2 \quad (2.8)$$

In order to utilise this technique, a Java class within the ranking package was created to maintain the multidimensional arrays to house the matrices mentioned above with methods to implement the necessary computations. These methods were used between every three iterations to accelerate convergence.

2.2.2.2-Separation of minimum rank value

Whilst Quadratic Extrapolation aided somewhat with computation time for very large graphs, said computation was still taking multiple days to finish. However, remembering the idea of the minimum rank value that a node gains from universal links, links due to the damping factor or due to even distribution from empty nodes, a lot of the repetitive operations within each iteration can be curtailed.

In each iteration, every node is given a base score derived from the universal links. These links are uniform to every node. Thus, if in every iteration the numerical values pertinent to calculating the minimum value for the proceeding iteration are stored, the process can become vastly more streamlined. This, and a conversion from calculations based on the outlinks flat matrix to its twin, the inlinks flat matrix allows vast orders of improvement in terms of ranking computation time. The new process for each iteration *k* is as follows. For every node *i*, the minimum value min_k derived from iteration *k-1* is its base ranking score value. On top of this, for every inlinking node *j* that node *i* has (if any) the following value is is added

$$d/o_j * R_{k-1}(j) \quad (2.9)$$

where *d* is the damping factor, o_j is the number of outlinks *j* has (derived easily from the outlinks flat matrix) and $R_{k-1}(j)$ is the ranking score computed for *j* in iteration *k-1* (the previous iteration). If *i* has no inlinks, its ranking score simply equals min_k.

To allow the calculation of the minimum score for the proceeding iteration, after the value of the ranking score for every node *i* is calculated, if it has no outlinks (and so it's vote must be evenly distributed over all other nodes) its value is added to $dead_k$, initialised as zero at the start of every iteration. If node *i* does indeed have outlinks, its score is instead added to $live_k$ also initialised to zero at the start of each iteration. After iteration *k* is completed, the value of min_{k+1} can be calculated using the following formula:

$$min_{k+1} = (dead_k / n) + (live_k * (d/n)) \qquad (2.10)$$

Finally, vast improvements in computation time were being observed. The CiteSeer NTriples graph file could be parsed, the various data entities in the ranking package filled and rankings output within twenty seconds. The actual ranking of the in-memory graph took approx. 2 seconds to reach an L1 residual value of less than 0.1, in about 15 iterations.

2.2.3-Weighting Feature

So far all the techniques described have been for analysing graphs where links are unique and one node cannot link another more than once. Intuitively in some domains however, a counting system for the number of times one node links another could be useful. It is perhaps not useful with regards the graph of research papers citing one another duplicate citations from one paper to another is not encouraged. Similarly it is not applicable to the FOAF or Patent data graphs. It may however be applicable to the graph of authors co-authoring with one another, derived from the CiteSeer dataset.

If author A co-authors 10 papers with author B, and co-authors 4 papers with author C, it could be said that author A is putting more of a vote of confidence in author B, with a weighting of 10/4 as compared to author C. Thus it is possible to derive weightings for the graph, and such weightings should allow better results than normal according to this consideration.

To implement the weighting feature in the ranking package, a parallel count flat matrix to the flat outlinks matrix was programmed. For each index in each row that corresponded to a link within the outlinks matrix, a corresponding entry was created in the count matrix to reflect the number of that particular link that was encountered. These count elements were then used as ratios in which to divide the nodes "vote". Unfortunately however, due to issues with the CiteSeer dataset, this weighting procedure was never employed.

2.2.4-Towards a search engine component

At this point we now have a ranking package featuring nine classes. These classes include a parent class `FlatMatrix`, which is extended by the classes `ConnectivityMatrix` and `CountMatrix`. Also within the package are two classes to provide lookups of node identifiers to node indexes, namely `NamesVector` and `NamesHashTable`. There are then three utility classes, one is a simple Pair which represents a node identifier/ rank score pair and has a comparable method to facilitate sorting with a Java `Collection` object, the second is the `QuadraticExtrapolation` class and the final is `RankingConsole` which features the main method which to this point was solely concerned with command line arguments and file I/O.

Whilst almost all the necessary backend components are in place, an interface through which the ranking component can be accessed by a Web agent would be necessary for it to be integrated into a search engine architecture. Also, the dynamic ranking techniques, namely local ranking and clustering would need to be implemented. And so it was.

2.2.4.1-Providing a HTTP interface

A Java servlet was created to provide the HTTP interface through which specific ranking requests could be passed to the ranking package, more specifically to an instantiation of a `RankingConsole`. The servlet allows requests to be issued to it via HTTP POST or GET. To try and make the architecture of the search engine more efficient, a keyword index was also instantiated by the servlet. This is the topic of *Section 3.2*.

The class `RankingConsole` was reconfigured so as to allow instantiation of it by a Java servlet. Upon initialisation, the `RankingConsole` class reads in an input file which comprises of the entire graph data in NTriples format for the given dataset. It parses this data as normal and loads the relevant data entities in the class. This all happens when the servlet's init method is called, and once initialised, the servlet maintains a reference to the `RankingConsole` instance as a class variable. The initialisation takes about twenty seconds and is called the first time the servlet is issued a request after being deployed.

The servlet can be passed a YARS URI (more in *Section 3.1*) when a semantic query is requested by the user. The servlet connects to YARS via a HTTP GET request, parses out the returned subject identifiers and passes them onto the `RankingConsole` to be ordered according to the global ranking scores determined upon initialisation.

The servlet can also be passed a keyword query (more in *Section 3.2*) as requested by the user. The servlet then uses the keyword index it references to return a list of subject uris that have a keyword hit. The same process can be then applied again passing the list of subjects to be ordered by the `RankingConsole`.

Once the `RankingConsole` has returned a list of ordered results, the servlet begins outputting them in order via its `OutputStream`. It usually takes 1 or 2 seconds for the servlet to return results from such a response for results sets of less than 10,000. Above that response time will not usually exceed 5 seconds.

The `RankingConsole` can also receive requests from the servlet for dynamic ranking services i.e. local ranking and clustering.

2.2.4.2-Dynamic Ranking Capabilities

The servlet can request local ranking or clustering services from the `RankingConsole` instantiation it references. If the user specifies a desire for such a service, this request is passed on to the `RankingConsole` object to be processed.

In order to allow such processing, the `RankingConsole` class has been equipped with an overridden init method which allows the creation of a new `RankingConsole` class that encapsulates a subgraph of the factory object. This is done via a specific method in `RankingConsole` which is passed an argument of a list of resource identifiers returned from either the user's keyword or semantic query (more in *Chapter 3*) which has been processed by the servlet. This list of resources is the subgraph which needs to be returned.
In order to create the subgraph, a blank `RankingConsole` object is initialised using the

overridden init method. The `RankingConsole` class then uses its `NamesHashTable` to do lookups of all the original indexes from the main graph of all resources passed to the method. These are all indexed and the string identifiers of the resources are cached in the new `RankingConsole` object for the subgraph.

If clustering is required, the class will take the user requested parameters from the servlet and perform clustering based on those parameters which are direction of clustering and number of hops. Depending on which direction of clustering (or both) is required, the method will fill the new subgraph instance of the class with all of the resources linked by those nodes already in the subgraph, as indicated by the inlinks and outlinks flat matrices from the main graph. This is done iteratively for the number of hops required.

Once the string identifiers for the desired nodes of the subgraph have been loaded into the new `RankingConsole` instance, the twin flat matrices (and optional count matrix) must be built. Essentially, using the equivalent main graph matrix, any links that exist between nodes indexed in the subgraph are added to a given matrix. The ranks are then computed and returned to the servlet to be output.

Finally we have an application which will allow both global and dynamic ranking of resources indexed, and which is suitable for use as a search engine component.

Chapter 3: Data Indexing Tools

In order to prepare and index the three datasets pertinent to the work portrayed in this thesis, various Semantic Web technologies and languages were applied. By describing the datasets in RDF, an indexing tool called *YARS*[8] could be used on the data [5].

Unfortunately, though YARS allows the specification of very expressive semantic queries, the version used for this project did not feature keyword indexing. A separate tool called Lucene[9] was used to index the keywords found in the datasets.

3.1-YARS

YARS or *Yet Another RDF Store*, is a Semantic Web application which allows the storing and indexing of data described in RDF-NTriples. YARS features a *REST-ful* HTTP interface which allows data to be issued for storage via HTTP PUT, and can be queried via HTTP GET.

YARS is implemented in Java and is packaged within a WAR file intended for deployment in an Apache Tomcat environment.

YARS uses optimised index stores to allow fast retrieval times. It centres around six different indexes which are accessed depending on the characteristic of the query. Each store allows optimised lookups of subject, predicate, object and/or context given a query.

Queries passed to YARS are in the N3QL format, an N3 based query language. N3QL allows advanced "semantic" queries to be issued. These queries are very flexible and can be quite useful if, for instance, you intend looking for resources (subjects) with specific property (predicate) values (objects). In fact you can issue queries based on knowledge of many different combinations of these three components. You can also determine the data returned, by specifying its desired source (context). Once you have specified the properties of the data you require, you can specify exactly what you want in the results set. YARS will then promptly begin returning results.

[8] http://sw.deri.org/2004/06/yars/
[9] http://lucene.apache.org/java/

If you remember the N-Triple snippet in *Fig.1.2*, if we were to load this into a YARS instance, and then enter the N3QL query of *Fig.3.1* into a HTML GET request for YARS, the result set, *Fig. 3.1*, would be all the triples which have subject URI `<http://www.ppl.com/Tim>`.

```
@prefix : <http://sw.deri.org/2004/06/yars#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ql: <http://www.w3.org/2004/12/ql#> .

<> ql:select {
      ?s ?p ?o .
}; ql:where {
       ?s rdfs:type <http://www.animals.com/Person> .
       ?s <http://www.verbs.com/authored> “Hamlet” .
       ?s ?p ?o.
} .
```

Figure 3.1 Example N3QL query

The query can be broken down into three constructs. The first section of the query allows the definition of *local-prefixes* for the namespaces used in later sections. Local prefixes are merely creations of convenience, as they allow a user specifying a query with the same namespace used multiple times to use the prefix and the property instead of the full URI. For instance, in the query illustrated in *Fig. 3.1*, the local prefix `rdfs:` is defined as the URI `http://www.w3.org/2000/01/rdf-schema#` and so instead of having to write out the full URIs of properties which use the namespace, such as `type` one can use `rdfs:type` to replace `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`.

The second component of the query, defined by the `ql:select` construct allows the indication of which variables defined in the `ql:where` section or other resources should be returned from the query. Selected in *Fig. 3.1* are the variables `?s ?p ?o .` which refer to subject, predicate and object respectively as we will see in the `ql:where` portion of the query.

The `ql:where` is the final, and probably most important element of the query. It allows the definition of the properties of any of the variables which appear in the `ql:select` portion of the query. Any variable in `ql:select` must have its properties defined in `ql:where`. In *Fig.*

3.1, the variables `?s ?p` and `?o` are defined by three lines in the `ql:where` element. The first line restricts the `?s` variable to all subjects whom have a property `rdfs:type` equal to the URI `<http://www.animals.com/Person>.` The second line further restricts the `?s` variable to a subject which also has also a property defined by the URI `<http://www.verbs.com/authored>` equal to the literal `“Hamlet”`. Now that the `?s` variable has been suitably bounded, the third line defines `?p` and `?o` as all of the properties and values thereof for all subjects matching the specifications defined for the `?s` variable.

Given an input dataset of *Fig 1.2* into YARS, *Fig 3.2* indicates the data that is returned by YARS given query *Fig 3.1*.

```
<http://www.ppl.com/Tim> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.animals.com/Person> .
<http://www.ppl.com/Tim> <http://www.generic.com/name> “Tim” .
<http://www.ppl.com/Tim>  <http://www.humanproperties.com/is_gender> <http://www.genders.com/female> .
<http://www.ppl.com/Tim> <http://www.verbs.com/authored> “Hamlet” .
<http://www.ppl.com/Tim> <http://www.ownership.com/owns> <http://www.move.com/Toyota> .
```

Fig. 3.2 Results from N3QL query in Fig 3.1

Such advanced queries were used to isolate the NTriples graph data from their respective datasets by issuing a query to return all triples with the property which equates to the links in the graph structure (e.g. the triple `?s foaf:knows ?o .` entered into the select and where components of the query and the local prefix `foaf` defined). These files would be eventually used as the input file to initialise the global instances of `RankingConsole` classes for the different datasets.

This idea of semantic querying is also offered to the user as an alterative to a keyword search. Where the user knows the exact properties of the resources (s)he wants returned in the results set, these properties can be specified by a dynamically created search tool, created specifically for each dataset .

The same basic principles are applied to the indexing of very the large datasets as are to the examples provided. The response time of YARS for queries is quite remarkable; however the creation of the indexes it uses internally can be quite slow. With this in mind, as YARS is loaded it only creates one index called *SPOC*, which indexes triples in their natural order.

This will only return results where the subject is the provided entity. To fully exploit the advanced query capabilities of YARS, one must build the other indexes one by one through a HTTP GET request. Each one can take days to fully build for large numbers of triples in the dataset. If one were then to go back and add more data to YARS and wanted that data to be fully indexed, all indexes besides the SPOC index would have to be rebuilt. These time constraints, and others explained later, pressed heavily on the desired timeline of this project. However, to reiterate, once the data is loaded into YARS and fully indexed, its performance is generally quite remarkable, even for indexing large datasets, conforming to the old adage, *good things come to those who wait.*

3.2-Lucene

Apache Lucene is an open source project implementing a keyword index, which is a type of inverted index. An inverted index is one which reverses the natural process of indexing. Therefore instead of a resource indexing keywords in the form of for example, a title, a description, a list of interests etc. these keywords can be used to index the resources they appear in.

This is the basic premise behind keyword queries. A user can give a word or phrase which they wish to appear somewhere in the resource they require and the keyword index will return hits for that query. However, Lucene also offers quite advanced keyword searches, not just limited to single keywords, exact phrase match or Boolean type properties. Such advanced keyword searches include wildcard entries, fuzzy searches, proximity searches etc[10].

Lucene is implemented in a Java package which is ideal for integration into the project. It offers classes to aid in the creation of an on-disk index, and classes to aid in the querying of such indices. The creation of the index is only required once and can be pre-computed. The querying is computed at run-time when a user issues a request. The obvious course of action to take was to build the Lucene index simultaneously with the YARS datastores.

To build a practical keyword index, only the objects of triples with certain properties were

[10] http://lucene.apache.org/java/docs/queryparsersyntax.html

indexed. Such properties were selected for each dataset based on the amount of text that would appear in that object, and whether or not a user might wish to search via keywords from that property. The objects of all such triples were of course literals, or string values. No property which would have a URI or blank node as object were selected. These keywords were then directly indexed against the subject identifier in the triple. Different properties were not differentiated, and all keywords were all conglomerated into one index. In hindsight, it may have proved purposeful to provide different fields for each property type the keywords were sourced from. This would allow specific field search by the user.

Once such an index was created for a datastore, the classes used to query it were embedded in the ranking component. The reasoning behind this is discussed in *Chapter 6*.

Chapter 4: Preparation and Indexing of the Datasets

Three datasets were chosen as use-case examples for the tool which this project aspires to create. All three datasets are very different in many aspects, they all require different methods of acquisition, they are all described in different metadata formats and they describe very different resources. One factor that link all three are their inherent "simple" graph characteristics. This Chapter proceeds to describe the acquisition, preparation and indexing of each of the three datasets

4.1-CiteSeer Computer and Information Science Publications Dataset

As already described in *Section 1.3.1*, CiteSeer is an archive of computer and information science metadata. This metadata is of the format OAI and is available as a single compressed archive.

4.1.1-Dataset Acquisition

The acquisition of this dataset was perhaps the most straightforward of all three. The entire dataset is available in one compressed archive with about 570 files indexing 1,000 publications each. Data acquisition basically involved downloading the 378 MB file and uncompressing it on a Linux server into a directory. All the files were then directly accessible by a tool created to convert and index the data.

4.1.2-Preparation for Indexing

YARS indexes RDF-NTriples, and so this was the destination metadata format for the files. As we will see however, the various RDF languages are easily inter-convertible. The original files are described in OAI which is quite similar to XML. Therefore if we first convert from OAI to XML, then XML to RDF-XML and finally RDF-XML to RDF-Ntriples, we will have the data in the necessary data format.

The conversion from OAI to XML was exceedingly simple. The essential difference between both is the dearth in OAI of a root element, which is a mandatory component of an XML document. To counteract this, after each file had been read into a buffer, a start tag and an end tag were appended to the buffer. The major stepping stone in the data conversion path lay directly ahead: converting from plain XML to RDF-XML.

Although the suffix of RDF-XML is XML, they are two wholly different breeds of metadata format. The first step in the conversion process was to create an RDF specification to which the XML documents could be converted. The RDF specification would have to encapsulate all the relevant data from the XML format in the specific way that RDF requires and that makes RDF interoperable.

The main difference between RDF-XML and plain XML is that plain XML need not define URIs for its elements, nor need it actually make any division between entities and properties. Referring to the example XML document in *Fig 1.1*, there does not exist any property between the entities `<products>` and `<games>`. There is no computer interpretable relation between the two physical entities. Such an absence of structure stops computers from abstracting the semantics of the data.

In an effort to remain true to interoperability, existing namespaces were used as much as possible as mappings from the XML data. One slight advantage of this dataset for such a conversion was the widespread use of elements from the Dublin Core namespace[11] (traditionally given the local prefix `dc`), which are not only relevant to plain XML applications but are also valid RDF. These elements were preserved.

Another existing namespace used, but not already present in the data, was the FOAF namespace used for describing people. This was used mainly to map metadata about authors where possible.

Despite this attempt to stick to existing namespaces, a new specification was necessary to map certain elements. This specification was created and given the namespace `http://sw.deri.org/2005/07/CiteSeer/`. The local prefix `citeseer` will be adopted to represent this URI from herein.

The finished specification for mapping to from the XML metadata featured two classes of resources, authors denoted by the `foaf:person` element from the FOAF namespace and papers denoted by the a new element, `citeseer:paper`. These resources have various

[11] http://dublincore.org/

applicable properties from existing namespaces and the new namespace. The main one of concern is applicable to resources of type `citeseer:paper`. It is the property `citeseer:cites` which allows one resource of type `citeseer:paper` to cite another resource of type `citeseer:paper`. This creates the graph which will be used to rank the papers.

Now that we have an idea of what elements from the XML documents we want mapped, and the structure of the RDF-XML document which is being mapped to, an *XSLT* document is used to carry out the actual mappings. XSLT is a transforming stylesheet which takes as input an XML document and can access data within that XML document using a technology called XPath. The XSLT can then fill this data into a new XML document or any other type of document. XSLT is itself an XML document that must be compiled. Java classes within the package `javax.xml.parsers` allow this process. Once this was implemented, the home stretch of the data conversion path lay ahead.

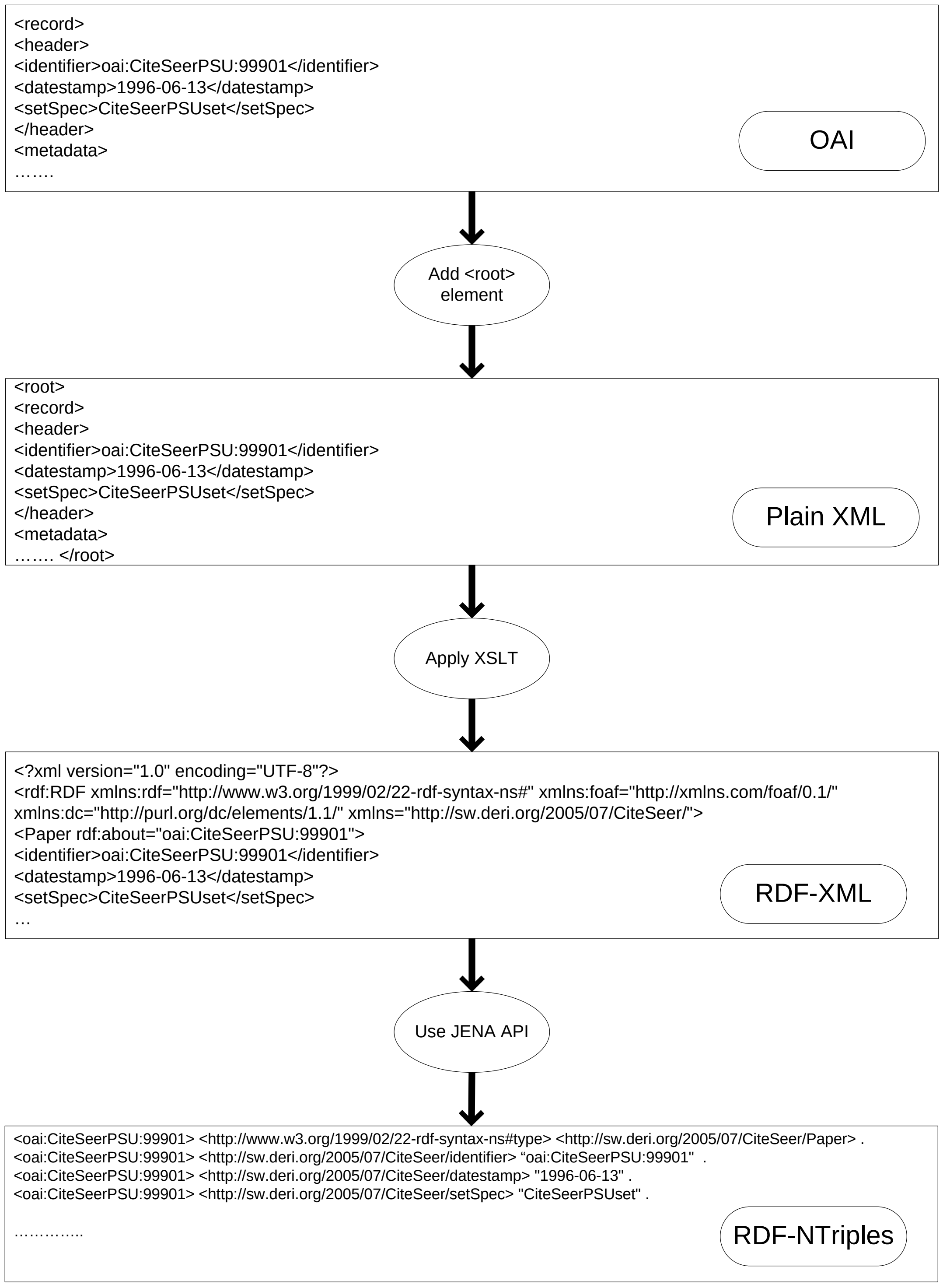
Conversion from one RDF language to another could be a daunting prospect. RDF-XML and RDF-NTriples aren't even remotely syntactically alike. Thankfully, yet another external open source Java package is available called Jena[12], an RDF API. It provides classes to provide conversions from one RDF language to another. *Fig. 4.1* illustrates the entire process of conversion from OAI to RDF-NTriples (also see Appendix A).

4.1.3-Indexing the Data

Now that the data is in N-Triples, it can be sent to a YARS via HTTP PUT. A new YARS instance was created and deployed in Tomcat. The entire dataset took numerous days to be converted and loaded. The auxiliary indices were also built. This would allow for semantic queries in the final application using the YARS instance as a backend.

Simultaneously to the indexing of the data, a Lucene index was maintained. The values of the Dublin Core elements `dc:title`, `dc:description` and `dc:subject` were chosen as properties which would generally have a value full of pertinent keywords.

[12] http://jena.sourceforge.net/



```
<record>
<header>
<identifier>oai:CiteSeerPSU:99901</identifier>
<datestamp>1996-06-13</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
</header>
<metadata>
…….
```

```
<root>
<record>
<header>
<identifier>oai:CiteSeerPSU:99901</identifier>
<datestamp>1996-06-13</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
</header>
<metadata>
……. </root>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="http://sw.deri.org/2005/07/CiteSeer/">
<Paper rdf:about="oai:CiteSeerPSU:99901">
<identifier>oai:CiteSeerPSU:99901</identifier>
<datestamp>1996-06-13</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
…
```

```
<oai:CiteSeerPSU:99901> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://sw.deri.org/2005/07/CiteSeer/Paper> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/identifier> “oai:CiteSeerPSU:99901"  .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/datestamp> "1996-06-13" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/setSpec> "CiteSeerPSUset" .

…………..
```

Figure 4.1 Path of Conversion from OAI to RDF-NTriples

4.1.3-Postprocessing and Issues with Dataset

Upon converting and analysing the data, some problems with the original dataset were revealed. In the original OAI format, a property `oai_citeseer:relation type="Is Referenced By"` was featured. This element was converted to `citeseer:referencedBy` the inverse property of `citeseer:references`. Unfortunately, this property was flagrantly inaccurate, for example, trying to purport that papers from 5 years ago were cited by papers from 20 years ago etc. In most cases, if one were to check a paper that another paper was supposedly referenced by, one would not find the presence of such a reference. The YARS version I was using did not feature delete capabilities and so the offending elements had to be left in (restarting the whole YARS indexing process would be too time consuming). However a new set of inverse properties were derived from swapping the subject and object of the `citeseer:references` triples, using a new element `citeseer:referenced_by`. These were sent to YARS.

Also the properties `oai_citeseer:relation type="References"` which were converted to the very significant properties `citeseer:references` were largely incomplete for many papers. Those that did exist however were reasonably reliable and so the work on ranking the links was continued.

Another issue in the dataset was garbled values for literals. In many instances, literals had values of nonsensical strings. The author metadata was particularly disappointing. Papers in the dataset are provided with unique identifiers such as `oai:CiteSeerPSU:99901`. Every time such an identifier is mentioned, it is unambiguous which paper is in question. No such identifiers exist for authors. A technique called *smushing* is used to provide consolidation of resources.

If one mention of an unidentified resource, in this case an author, shares a most likely unique property with another unidentified resource, both resources can be regarded as equivalent. This is by no means an ideal situation as different authors may have the same name, but for the best part it works well. However, in the original OAI database, the names of authors were given values such as “PhD” or “Masters”, sometimes given a title and other times not. Names of institutes and universities were also commonplace values for author names. This frustrated attempts at smushing the dataset and despite several efforts to resolve the issue by removing the offending author name values and cleaning up the other names, the ranking of authors would always be highly unreliable, and was abandoned. However, Paper resources were still rankable, although only with regards a subset of citations that actually exist, and so work continued.

4.2-GreatestJournal FOAF files

This dataset, introduced in *Section 1.3.2*. was perhaps the easiest to convert for indexing, but was probably the most awkward with regards acquisition.

4.2.1-Data Acquisition

This particular dataset was not available in a downloadable archive format, instead it was a graph of documents on the Web. When user A indicates that he is an acquaintance of user B, a link is placed in user A's FOAF file to user B's FOAF file. Thus, if a software agent were to visit user A's FOAF file and that software agent knew how to find links, such as the one to user B's FOAF file, then the software agent could apply the same technique to that FOAF file and continue on through the whole graph. This method is called crawling and is a common Web technique. Obviously not all FOAF files would be retrieved using this method as some FOAF files would not be linked by another user. They would not be important to the graph that would be analysed however for if no other user links to them, they would get a very low ranking score. GreatestJournal boasts 1.3 million users, many of which probably fall into the previous category. At time of writing, 325,400 FOAF files have been cached and indexed, but according to the ranking algorithm which gives an indication of how many unique nodes exist, only about 370,000 exist.

At first an existing crawler was used, but the crawler was overly complicated and somewhat buggy. Instead, a new crawler was built featuring just three classes. One class was used as a console for the crawler, defining a main method that allows the input of a seed file of links and also storing application data such as which links had already been visited (to prevent continuously revisiting the same pages). The second class extended Thread and did the actual FOAF file traversal, data conversion and upload to YARS. The third class was involved in opening input streams to the web pages. The console class calls a certain number of instances of the threaded class. The class is threaded so as to keep data available for processing at all times. This prevents a possible bottleneck caused by slow downloads of FOAF files.

4.2.2-Preparation and Indexing

Data conversion again was implemented by use of the Jena API. The FOAF files are already in RDF-XML so conversion to RDF-NTriples was trivial. There was one obstacle to a straight through conversion of the files however, and that was a lack of identifiers for the `foaf:Person` elements used to describe each user. This had to be manually handled by Java code written in the threaded class.

Whilst our previous attempt at smushing in the CiteSeer dataset proved unfruitful, one particular property of `foaf:Person` was ideal for smushing : `foaf:nick.` This had the value of the username of the person on GreatestJournal, and by simply appending this property's value onto a set URI, a perfect identifier is created. The property is so ideal for smushing because every time that the `foaf:Person` element appears in the dataset, so does the property. Also each `foaf:nick` value must be unique so there is no ambiguity possible and an added bonus, it cannot contain spaces or any characters that might make the new URI created difficult to handle.

With identifiers for every `foaf:Person` element appearing manually added, the modified RDF-XML code was converted by Jena and indexed in a separate instance of YARS to the CiteSeer data. Although the crawler has not at the time of writing yet managed to do a complete traversal of the graph of files, there still exists plenty of data to be indexed.

At one stage during the crawl, the YARS indices were built and the NTriples graph queried for. However many more files have been crawled since and because of time constraints, the NTriples graph and the Lucene index were created for the entire dataset as it stands by performing a dump of all data in YARS. This was completed by doing a complete dump of the YARS dataset which does not require lengthy index building computations. As the data is output from YARS, the triples with property `foaf:knows` were added to the NTriples graph file, and the triples with property `foaf:interests` were used in the creation of the Lucene keyword index. They were however mapped back to the `foaf:Person` element from hence it originated, as there exists a node of no specified type with in the dataset.

4.3-US Patent Metadata

This dataset, introduced in Section *1.3.3*, is potentially the largest of the datasets. It extends back as far as 1976, and features millions of metadata entries for both patent applications and granted patents in four different formats. The work described herein only applies to a subset of the data, all the patents starting from the first week of 2005.

4.3.1-Dataset Acquisition

The patent metadata is available for free download though an FTP server hosted by the US Patents Office. The files are weekly releases of zipped files for that weeks granted patents. In order to access the data, a crawler, was initially created to take an FTP directory and cache, unzip and index each file. However, after the crawler caches the data from the FTP site, once it flushes and closes the data stream, when the file is attempted to be re-opened for unzipping using classes from the `java.util.zip` package, the file is not recognised as a valid archive. This issue is sill unresolved, and so as to continue making progress with indexing the dataset, the files, 63 in all, were manually downloaded.

4.3.2-Preparation and Indexing

The metadata downloaded is encoded in plain XML, sticking to a structure which the Patent Office refers to as RedBook. By following similar steps as was employed to convert the CiteSeer archive, this data can be converted to RDF-Ntriples in a parallel fashion (see Appendix B).

The RedBook format is extremely expressive, with multitudes of layers of elements employed to create a patent description. If an attempt was made to preserve such a verbose description of every Patent, the user would be overwhelemed with seemingly meaningless properties for the resources in the results set. So as to avoid this problem, the specification to which the RedBook format file is mapped is a much more simplified, but still contains all the significant data relating to a Patent.

As of the time of writing, an official document outlining the specification has not been created, however, a new namespace is defined in the data at `http://sw.deri.org/2006/02/USPatent/` and the local prefix `pat:` will be used to refer to this namespace.

The specification features a single type of resource, which is of type `pat:Grant` (due to a mistake in an earlier version of the XSLT document, resources of type `pat:Patent` may also be found in the dataset, this should just be disregarded) which has a vocabulary of 21 properties to describe it. 19 of these properties are in the new namespace. This is because the elements used in the original documents are so specific that finding equivalent elements in existing namespaces becomes difficult.

Again, once the data is converted from plain XML to RDF-XML, Jena is used to convert the data on again to RDF-NTriples.

The keyword index was built from the values of triples which featured two of the properties which were used from the Dublin Core namespace, `dc:title` and `dc:description` .

Chapter 5: User Interface

For the purposes of a user search and retrieval tool, an application was required to offer the user the ability to issue the system either a keyword or semantic query, and present the user with a list of ranked results in a formatted easy comprehensible web page. The main issue here is formatting the results page. As the metadata for all the resources are stored in YARS, and YARS returns NTriples results for queries, the application would have to parse these NTriples results into a formatted HTML response page. Issuing the user with the NTriples results alone would not be acceptable as to a normal user ill-read on RDF description languages, such a results page would not be easily digested. In order to achieve this application, an existing tool was used as its core, called *Nodebrowser*[13].

5.1-Original Nodebrowser

The original application which was used as the core of the new user interface component was authored by Andreas Harth (a colleague) and myself and uses various pieces of code authored by Matteo Magni and Hannes Gassert (former colleagues). It is a WAR file, which when set the path to the YARS backend it services, can be deployed in Apache Tomcat. As already stated, this application uses YARS as a backend and takes a simple query in the form of an object. It then queries for resources with this specific object stored in YARS. Finally, it takes the NTriples results page returned by YARS, serialises it into RDF/XML and from this, generates a HTML response page. This application was intended for use where the YARS backend is filled with generic data, and is completely interoperable with any YARS instance.

One draw back of the original application was that it only offered one simple form of query, a query for resources with a specific object. This was very restrictive, and would not be enough for the system described herein.

Another draw back was that the application was ignorant of the schemas of the data available in YARS. To paraphrase, the original Nodebrowser application had no idea of the actual meaning of the URIs for the predicates and subject types stored in its YARS backend.

[13] http://sw.deri.org/2005/10/nodebrowser/

This makes it available for use with any YARS instance storing any schema or specification of data, as it was intended for.

One last drawback was that the application would not be interoperable with a ranking component as it featured no methods for interacting with said component. Not all datasets would feature graph traits and so ranking would not be portable to just any data store.

The metadata our three instances of YARS have been filled with have a known structure and schema and are rankable. For two of the datasets an XSLT stylesheet was used to convert the data using specific namespaces and specifications, and so the schema is already well known. For the third dataset (GreatestJournal), the only specification used is that of the FOAF namespace. The data is also computer generated and so it all has an identical method of creation, and so the properties of the schema for this dataset are also easily derived. By exploiting this prior knowledge of the structure of the data a much more advanced and user friendly tool could be created.

5.2-New Features Added

5.2.1-Lucid HTML Serialisation

The first, and most potent new feature added was to allow a more human digestible HTML serialisation of the NTriples to take place. The first step was to make a HTML serialisation class, which reads in the welcome page of the application, a HTML file, and finds a tag within the file, specifically placed there to indicate where results should be displayed (see *Appendix D.1*). Essentially it uses the welcome page html code as a template for building the results page. To implement a more aesthetically and comprehensible results page format, the original application was also fitted with the ability to read in a *config.xml* file (see *Appendix C*).

This configuration file is created with the specific schema in question in mind, and allows the configuration of various serialisations of different properties (predicates) and different subject types (as defined by the `rdfs:type` predicate) to be specified. This would allow conversion back from predicate and subject type URIs into words or phrases (a reversal of the creation of the NTriples snippet in *Fig 1.2*).

Each predicate found within the YARS datastore to which the application is a frontend, must have its URI listed and be given two additional entries, a human readable label and a priority or order. In the HTML serialisation process, the URI of the predicate parsed is replaced by the label and the predicates for a resource are listed according to the order specified. The motive behind this is that certain properties of resources are more significant than others. For instance, for the HTML serialisation of each resource, what that resource is should appear first (is it a paper or an author), then some form of title or name for the resource, then perhaps some form of description or maybe date, etc. To allow this, each predicate is given an element in the config file, and each predicate element must have properties denoted by `<uri>`, `<label>` and `<order>` tags. The lower the order number, the higher up the predicate it refers to is serialised.

Two optional elements are also offered for each predicate. The first is the `<ignore>` tag which allows the predicate to be ignored by the HTML serialisation and thus be skipped when the results page is displayed if its value is `true`. This can be used where certain predicates of a resource are not helpful to a user. An instance of when this was used was to hide the `citeseer:refererecedBy` property converted directly over from the equivalent erroneous property in the CiteSeer dataset. If not specified it defaults to the value `false`.

The second tag, `<internal>`, can have three values; -1, 0 or 1. If not specified, it defaults to the value -1. It refers to the type of object that such a predicate would have, and how it should be handled by the application. Some properties refer to HTML pages (e.g. foaf:homepage) and so the objects of these predicates would be serialised as a HTML hyperlink. This is signified as the value 0. Other properties may be plain text, and will be treated as such (e.g. `dc:title` which would be given the value -1). Finally, some properties may have URIs that refer to resources indexed by the datastore, for instance `citeseer:references` would have as object a URI of a resource within the dataset, another paper. When this is being handled, it offers a link which tells the browser application to show the data relating to that file. Such properties are given the value 1. Internal links in the results page are displayed in maroon, external links open in a new browser window and are blue.

Subject types must also be listed with their URIs, a label and also a stylesheet class. Each of these are specified by `<type>, <typelabel>` and `<divclass>` respectively. The type of a resource (e.g. paper, author etc.) is specified as a URI which is the object of the `rdfs:type` label. When this is encountered, in the serialisation of the resource, the label is inserted into the heading to identify what is being described. The `divclass` allows an aesthetic amelioration in the results page by defining different stylesheet labels for different types of resource. These labels must also be reflected in the stylesheet, where different stylesheet classes can be defined for division resources are displayed in.

This configuration file is parsed and loaded into class variables by a class included in the application package. The file is parsed upon initialisation of the class (which is passed the path of the config.xml file) and a reference to the class is passed to the HTML serialisation methods defined in a separate graph.

5.2.2- Interface with Ranking: Buffered Results, Navigation and Sessions

The original application offered a very simple functionality, give me an object and I'll return you a HTML page with all of the results. The interaction with YARS was quite simple, one query, one results page. However, such simplicity goes out the window when one adapts the application to interface with the ranking component.

The ranking component returns a list of resources in order of highest rank first. Each resource must then be wrapped in an N3QL query and sent to YARS. If the results set included thousands of results, the application would overload.

In order to allow speedy response times, one would have to buffer results, and only query YARS as necessary. To accomplish this in the application, one simple salient fact can be pointed to; a user cannot read tens of thousands of results at one time. A limit of ten results per page was retrofitted into the application.

Buffered results were introduced. When a user enters a query, a request is sent to the ranking servlet and the number of results in the response is counted. The first page is then displayed with the number of results found and the first ten resources serialised. That is to say only ten queries are sent to YARS. A next option is offered and upon clicking, the next ten results in order of how they were ranked are displayed, etc.

This was in fact quite difficult to implement in the application. Offering only forward browsing to a user would be easy, whereby only a next option to load the next 10 results would be offered; it is the option to revisit a page which makes implementation awkward. The application had to store and cache all results as they were browsed and maintain alertness of the current state of browsing. Navigation was then provided on the results page.

One issue this raised was the problem of multiple users accessing the application simultaneously. To counteract this problem, sessions are used. Sessions are a feature provided for in Java servlet development to provide specific users with their own copy of data, or a means of referencing each users own application state. This means that when each user logs into our system, a new session starts and each user is provided with their own data cache of previous results, and their own navigation system.

One unresolved problem which the application has is when a user uses their web browser (external) navigation and not the internal navigation (i.e. click the Back/Forward button on their Internet Explorer of Firefox application). This confuses the application as to where in the cache of data the current results being displayed are situated. If the internal navigation is used again, the application fails to successfully complete the task required. A new query is required to restart the data cache. Thus, it is imperative only to use the internal navigation provided on the results page.

5.2.3- In Tab Browsing

This was another feature added to the new application which essentially offers users the ability to get the details of a resource which appears as an internal link (because of a predicate with the `internal` set to 1, e.g. `citeseer:references`), within the current results page by clicking on a + displayed to the left of its link. This expands the resource with a new division displaying its details instead of its link, but in the same location as the link was in the results page. If any internal links appear in the new division, they too can be expanded again. This allows effortless browsing of the graph of resources, allowing tree type browsing. Once finished with these expanded results, they can be returned to their normal state again by clicking on the – link provided, which can be done at any level of an expanded tree of results. See *Appendix D.2* and *D.4* for screenshots.

5.2.4- Advanced Semantic Searching

The final addition to the original application was to allow the user to specify a semantic search, or a search which allows the user to specify the exact properties that a resource must meet before it is returned. It is not as such viable to expect the user to have knowledge of N3QL or some other advance query language in which to express their requirements, however using the config.xml file, the application knows all possible subject types and predicates. Thus the application can dynamically create an advanced semantic search form, which offers users a choice of possible subject types and properties in drop down HTML boxes, and the desired values of those properties to be entered into a text field, as reflects the resources they desire. Once submitted these parameters are then encoded by the application and used to create an N3QL query for YARS. See *Appendix D.3* for a screenshot of the semantic searching form.

Chapter 6: Search Engine System

With an efficient algorithm package (with HTTP interface) for computing ranking scores and dynamic ranking values given an NTriple file as input, an optimized means of indexing RDF in NTriples, three datasets with a means of conversion to NTriples, three keyword indexes and an advanced user interface tool, creating the final search engine systems for the three datasets is the equivalent of a jigsaw puzzle. Three such systems were created, one for each dataset. Each system requires three components, each with a HTTP interface and deployable within a Tomcat Apache environment, and so in total, nine components were deployed on nine different servers, allowing a Web Services type architecture to be employed.

6.1-System Architecture

The system architecture reflects a Web Services type coordination where each main service of the system is isolated as a single component with its own interface (the exception is ranking and keyword which are conglomerated for efficiency). Once the datastore and keyword index have been filled for a dataset, and the NTriple graph data isolated and provided to the ranking component, the system has all the prerequisites it needs and can be built on top of it.

Fig. 6.1 shows said architecture, with all of the involved components, the dataflow between them. It also shows the partition between runtime operations and pre-computed operations necessary. All of the runtime operations are instantiated by a user request. All other operations are done in setting the system up. All runtime dataflows are through HTTP methods, where requests are a HTTP GET or POST and all responses are formatted results page which are parsed by the receiving component.

Differentiated between, in *Fig. 6.1*, are the requests and responses for a semantic query and a keyword query, as desired by a user. Shown in red are these for a keyword query, and shown in blue are for a semantic query. As the keyword index is within the same component as the ranking algorithm, the ranking algorithm can get a list of hits for a keyword query internally in the component. However for a semantic query, the component must send a request to YARS to get a list of all hits for the query before it can rank them.

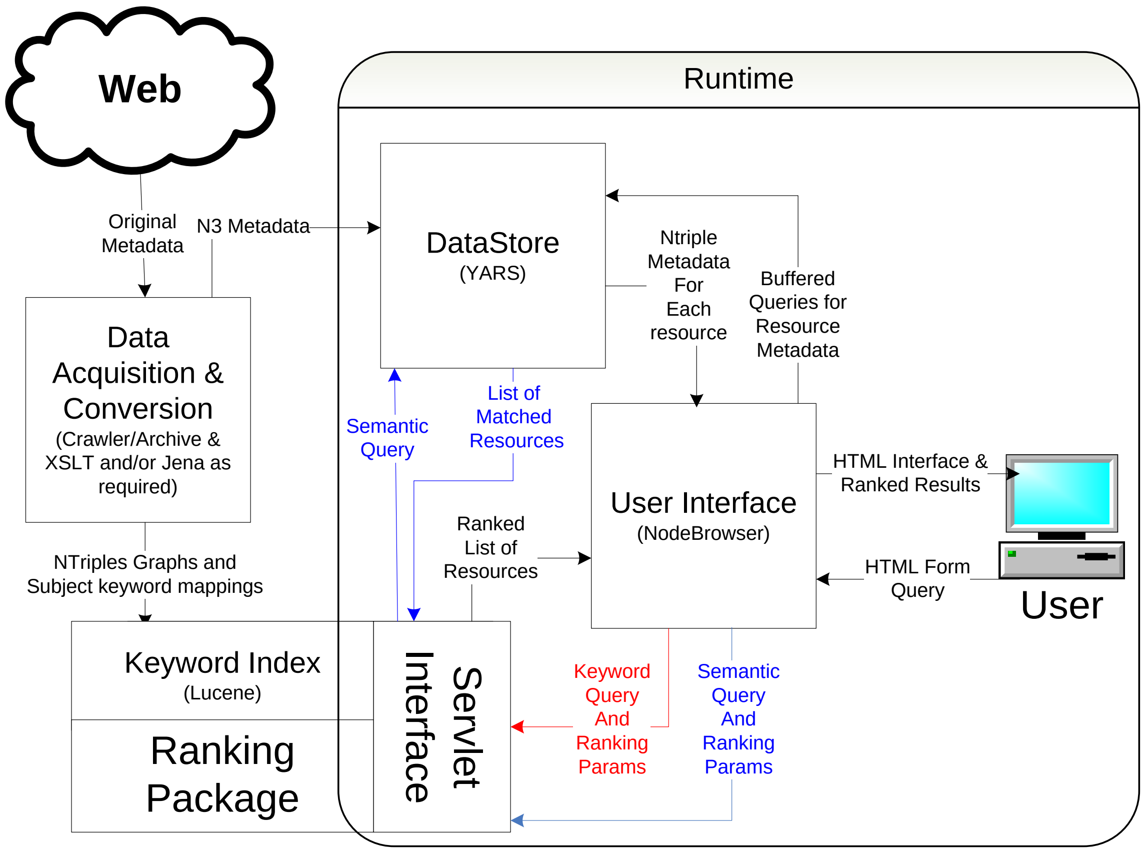

Web
Runtime
Original Metadata
N3 Metadata
DataStore (YARS)
Ntriple Metadata For Each resource
Buffered Queries for Resource Metadata
Data Acquisition & Conversion (Crawler/Archive & XSLT and/or Jena as required)
Semantic Query
List of Matched Resources
User Interface (NodeBrowser)
HTML Interface & Ranked Results
HTML Form Query
User
Ranked List of Resources
NTriples Graphs and Subject keyword mappings
Keyword Index (Lucene)
Ranking Package
Servlet Interface
Keyword Query And Ranking Params
Semantic Query And Ranking Params

Figure 6.1 Architecture of the search engine system

6.2-Runtime Operation

To path taken for a user specified query is as follows, the user interface sends either the semantic query (in the form of a YARS URI with the newly created N3QL query encoded and appended onto it) or the keyword query to the ranking and keyword index component. It also sends the ranking parameters (clustering and local ranking values). This component then builds a list of hits for either the keyword query (completed internally) or the semantic query (sent to YARS) and ranks them according to the parameters received. It creates a specially formatted response page of ordered results, which the user interface parses. Now that the user interface component knows which resources to retrieve first, it begins building YARS queries, one for each resource and ten at a time, to YARS requesting the data associated with that resource. YARS returns an NTriples results page for each query which is then parsed and serialised to HTML by the user interface component.

6.3-Final Implementation and Performance

Final implementation of the above architecture was implemented for the three architectures. As previously stated the three main runtime components were deployed for the three datasets. Each implementation of the architecture proved to be successful in its intended operation.

For the CiteSeer dataset, there was about 300,000 nodes in the global ranking graph and about 1.3 million links (requirements for being a node in the graph were that a paper either cite or be cited by another). Those papers that were returned near the top of the results set were indeed important papers in the area bound by the query. This was easily verified by examining the `citeseer:referenced_by` property, and seeing that those near the top had been cited numerous times by other important papers. Trying to validate the results subjectively is quite difficult, and one must have faith in the idea of peer review which the algorithm is based on.

Dynamic ranking also proved a major success with response times for even large results sets being quite brief, less than two or three seconds generally.

For the GreatestJournal system, there was about 355,000 nodes present and a total of 5.4 million unique links. The number of links was initially of concern with regard performance issues, however this concern was devastated by empirical evidence that the architecture could handle the size of the graph. Indeed response times were again quite astoundingly low when one puts it in the perspective of the computational effort involved.

Again, for the patent dataset search engine, results were quite pleasing with response times also being very low. The number of nodes in the global graph was 1.6 million (many of which have yet to be downloaded into the system). The number of links was just over three million. Again response times were very pleasing, requiring no patience from the user.

Conclusion

The work described herein was in parts, very challenging but ultimately rewarding. A lot of programming expertise was required in Java to implement the described components of the architecture. Also, a firm grounding in Semantic Web technologies were of great aid in the pursuance of the work.

One major stumbling block in the project was the time taken for certain operations to successfully complete. These include obtaining, converting and indexing all of the data. Many several gigabytes of data was converted and analysed, cached and indexed. Such operations ran for days and made progress at times quite slow. This gravely affected the projected timeline of the work, and so some of the work has not been fully completed. However, all the design and application development is finished, and the tasks that remain, including YARS index building, a fully inclusive crawl of the GreatestJournal FOAF pages and finally, indexing of all patent data; all these tasks were put aside as they would take too long. With regards indexing the older patent data, much effort was put into creating a means of converting the arcane data format which they use, called GreenBook, into RDF. Unfortunately however, to date, this has not been used. Plans for future work do indeed include all these above tasks.

With regards building the YARS indexes, the purpose of this is to fully enable semantic searches, which currently are unavailable for the GreatestJournal and patent search engines, and have not been updated for the CiteSeer engine. These indexes involve essentially clicking an option on a web form, but take days to complete. Plans for future work also include this simple task.

Originally in the project, the ranking analysis techniques were intended to be pre-computed and entered into the datastore as a property of each resource. Dynamic ranking was not seen as a viable option, as it was presumed to be too slow and lead to response times for requests by users that would test their patience. It was a delight to see that the ranking was both scalable and fast, and response times for the architecture, even with runtime dynamic ranking are delightfully brisk.

The three applications provided are indeed quite user friendly and highly practical, and the results pages are easily browsed and read. In fact, I myself used the CiteSeer search engine for the purposes of literature review and research for the project.

Another noteworthy fact is that the system is highly portable. Once data is in RDF and has some graph inherent in it, the above described architecture is applicable. Hopefully practical use of the application will be extended to other datasets and areas, and colleagues of mine have already expressed interest in implementing such a system for their own data.

All in all, the work presented here in was a (somewhat unexpected) success, in particular with regard to response times for queries, and introduces a new means of bring together two research areas, the ever nascent area of the Semantic Web, and also the area of ranking algorithms and PageRank.

Bibliography and References

[1] S. Decker *et al.*, “The Semantic Web: The Roles of XML and RDF”, IEEE Internet Computing, (Sep./Oct. 2000).

[2] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. “The Pagerank Citation Ranking: Bringing order to the web.” Technical report, Stanford Digital Library Technologies Project, 1998.

[3] T.H. Haveliwala. “Topic-sensitive PageRank”. In Proceedings of the Eleventh International World Wide Web Conference, 2002

[4] S. D. Kamvar et al. “Extrapolation Methods for Accelerating PageRank Computation”. In Proceedings of the Twelfth International World Wide Web Conference, 2003

[5] A. Harth, S. Decker. “Optimized Index Structures for Querying RDF from the Web”, Proceedings of the 3rd Latin American Web Congress, Argentina, October 2005.

APPENDICES

A: CITESEER OAI TO RDF-NTRIPLES CONVERSION

A.1: XSLT USED

```
<?xml version="1.0"?>
<!DOCTYPE stylesheet [
<!ENTITY cr "<xsl:text>
</xsl:text>">
]>

<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:oai_citeseer="http://copper.ist.psu.edu/oai/oai_citeseer/"
  xmlns="http://sw.deri.org/2005/07/CiteSeer/"
  exclude-result-prefixes="content oai_citeseer"
  version="1.0">

<xsl:output indent="yes" cdata-section-elements="content:encoded" />


<!-- general element conversions -->

<xsl:template match="/">
  <rdf:RDF>
    <xsl:apply-templates/>
  </rdf:RDF>
</xsl:template>

<xsl:template match="record">
  <Paper rdf:about="{header/identifier}">
  <xsl:apply-templates />
  </Paper>
</xsl:template>

<xsl:template match="dc:creator|creator">
&cr;
  <dc:creator><xsl:value-of select="." /></dc:creator>
</xsl:template>

<xsl:template match="dc:rights|rights">
&cr;
  <dc:rights><xsl:value-of select="." /></dc:rights>
</xsl:template>

<xsl:template match="dc:date|date">
&cr;
  <dc:date><xsl:value-of select="." /></dc:date>
</xsl:template>

<xsl:template match="dc:subject|subject">
&cr;
  <dc:subject><xsl:value-of select="." /></dc:subject>
</xsl:template>

<xsl:template match="dc:format|format">
&cr;
  <dc:format><xsl:value-of select="." /></dc:format>
</xsl:template>

<xsl:template match="address">
&cr;
  <address><xsl:value-of select="." /></address>
</xsl:template>

<xsl:template match="dc:source|source">
&cr;
  <dc:source rdf:resource= "{.}" />
</xsl:template>

<xsl:template match="datestamp">
&cr;
```

```
  <datestamp><xsl:value-of select="." /></datestamp>
</xsl:template>

<xsl:template match="dc:language|language">
&cr;
  <dc:language><xsl:value-of select="." /></dc:language>
</xsl:template>

<xsl:template match="title|dc:title">
&cr;
  <dc:title><xsl:value-of select="." /></dc:title>
</xsl:template>

<xsl:template match="description|dc:description">
&cr;
  <dc:description>
  <xsl:value-of select="." />
  </dc:description>
</xsl:template>

<xsl:template match="contributor|dc:contributor">
&cr;
  <dc:contributor><xsl:value-of select="." /></dc:contributor>
</xsl:template>

<xsl:template match="identifier">
&cr;
  <xsl:choose>
    <xsl:when test="starts-with(.,'http')">
      <dc:identifier rdf:resource= "{.}" />
    </xsl:when>
    <xsl:otherwise>
      <identifier><xsl:value-of select="." /></identifier>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="publisher|dc:publisher">
&cr;
  <dc:publisher><xsl:value-of select="." /></dc:publisher>
</xsl:template>

<xsl:template match="setSpec">
&cr;
  <setSpec><xsl:value-of select="." /></setSpec>
</xsl:template>

<xsl:template match="oai_citeseer:relation|relation">
&cr;
<xsl:choose>
  <xsl:when test="@type = 'References'">
    <references rdf:resource ="{oai_citeseer:uri|uri}" />
  </xsl:when>
  <xsl:when test="@type = 'Is Referenced By'">
    <referencedBy rdf:resource ="{oai_citeseer:uri|uri}" />
  </xsl:when>
</xsl:choose>
</xsl:template>


<xsl:template match="oai_citeseer:author|author">
&cr;
 <xsl:variable name = "name" select = 'translate(@name," ","_")' />
 <dc:creator><xsl:value-of select="@name" /></dc:creator>
 <authorPerson>
  <foaf:Person>
   <xsl:attribute name="rdf:about">
   <xsl:value-of select = 'concat("http://sw.deri.org/2005/07/CiteSeer/authors/",$name)' />
   </xsl:attribute>
   <foaf:name>
    <xsl:value-of select="@name" />
   </foaf:name>
   <affiliation>
    <xsl:value-of select="affiliation" />
   </affiliation>
   <authorOf>
            <xsl:attribute name="rdf:resource">
             <xsl:value-of select = '../../../header/identifier' />
            </xsl:attribute>
   </authorOf>
  </foaf:Person>
 </authorPerson>
</xsl:template>
</xsl:transform>
```

A.2:CONVERSION PATH OF A SINGLE CITESEER ENTRY

A.2.1:OAI (XML)

```
<record>
<header>
<identifier>oai:CiteSeerPSU:99901</identifier>
<datestamp>1996-06-13</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
</header>
<metadata>
<oai_citeseer:oai_citeseer xmlns:oai_citeseer="http://copper.ist.psu.edu/oai/oai_citeseer/" xmlns:dc
="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://copper.ist.psu.edu/oai/oai_citeseer/
http://copper.ist.psu.edu/oai/oai_citeseer.xsd ">
   <dc:title>Action selection in a hypothetical house robot: Using those RL numbers</dc:title>
   <oai_citeseer:author name="Mark Humphrys">
      <affiliation>University of Cambridge , Computer Laboratory</affiliation>
   </oai_citeseer:author>
   <dc:subject>Mark Humphrys Action selection in a hypothetical house robot: Using those RL
numbers</dc:subject>
   <dc:description>Reinforcement Learning (RL) methods, in contrast to
many forms of machine learning, build up value functions
for actions. That is, an agent not only knows
`what' it wants to do, it also knows `how much' it wants
to do it. Traditionally, the latter are used to produce
the former and are then ignored, since the agent is assumed
to act alone. But the latter numbers contain useful
information - they tell us how much the agent will
suffer if its action is not executed (perhaps not much).
They tell us which actions the agent can compromise
on and which it cannot. It is clear that many interesting
systems possess multiple parallel and conflicting
goals, all demanding attention, and none of which
can be fully satisfied except at the expense of others.
Animals are the prime example of such systems. In
[Humphrys, 1995], I introduced the W-learning algorithm,
showing one method of resolving competition
among behaviors automatically by reference to their
RL values. The scheme has the unusual featu...</dc:description>
   <dc:contributor>The Pennsylvania State University CiteSeer Archives</dc:contributor>
   <dc:publisher>unknown</dc:publisher>
   <dc:date>1996-06-13</dc:date>
   <dc:format>ps</dc:format>
   <dc:identifier>http://citeseer.ist.psu.edu/99901.html</dc:identifier>
   <dc:source>http://www.compapp.dcu.ie/~humphrys/PhD/../Publications/f.SOCO96.ps.gz</dc:source>
   <dc:language>en</dc:language>
   <oai_citeseer:relation type="References">
      <oai_citeseer:uri>oai:CiteSeerPSU:18816</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="References">
      <oai_citeseer:uri>oai:CiteSeerPSU:194839</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="References">
      <oai_citeseer:uri>oai:CiteSeerPSU:163604</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="References">
      <oai_citeseer:uri>oai:CiteSeerPSU:87435</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="References">
      <oai_citeseer:uri>oai:CiteSeerPSU:142710</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="Is Referenced By">
      <oai_citeseer:uri>oai:CiteSeerPSU:8115</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="Is Referenced By">
      <oai_citeseer:uri>oai:CiteSeerPSU:230110</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="Is Referenced By">
      <oai_citeseer:uri>oai:CiteSeerPSU:69071</oai_citeseer:uri>
   </oai_citeseer:relation>
   <oai_citeseer:relation type="Is Referenced By">
      <oai_citeseer:uri>oai:CiteSeerPSU:32645</oai_citeseer:uri>
   </oai_citeseer:relation>
   <dc:rights>unrestricted</dc:rights>
</oai_citeseer:oai_citeseer>
</metadata>
</record>
```

A.2.2:RDF-XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="http://sw.deri.org/2005/07/CiteSeer/">
<Paper rdf:about="oai:CiteSeerPSU:99901">
<identifier>oai:CiteSeerPSU:99901</identifier>
<datestamp>1996-06-13</datestamp>
<setSpec>CiteSeerPSUset</setSpec>
<dc:title>Action selection in a hypothetical house robot: Using those RL numbers</dc:title>
<dc:creator>Mark Humphrys</dc:creator>
<authorPerson>
<foaf:Person rdf:about="http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys">
<foaf:name>Mark Humphrys</foaf:name>
<affiliation>University of Cambridge , Computer Laboratory</affiliation>
<authorOf rdf:resource="oai:CiteSeerPSU:99901"/>
</foaf:Person>
</authorPerson>
<dc:subject>Mark Humphrys Action selection in a hypothetical house robot: Using those RL
numbers</dc:subject>
<dc:description>Reinforcement Learning (RL) methods, in contrast tomany forms of machine learning, build
up value functionsfor actions. That is, an agent not only knows`what' it wants to do, it also knows `how
much' it wantsto do it. Traditionally, the latter are used to producethe former and are then ignored,
since the agent is assumedto act alone. But the latter numbers contain usefulinformation - they tell us
how much the agent willsuffer if its action is not executed (perhaps not much).They tell us which actions
the agent can compromiseon and which it cannot. It is clear that many interestingsystems possess multiple
parallel and conflictinggoals, all demanding attention, and none of whichcan be fully satisfied except at
the expense of others.Animals are the prime example of such systems. In[Humphrys, 1995], I introduced the
W-learning algorithm,showing one method of resolving competitionamong behaviors automatically by reference
to theirRL values. The scheme has the unusual featu...</dc:description>
<dc:contributor>The Pennsylvania State University CiteSeer Archives</dc:contributor>
<dc:publisher>unknown</dc:publisher>
<dc:date>1996-06-13</dc:date>
<dc:format>ps</dc:format>
<dc:identifier rdf:resource="http://citeseer.ist.psu.edu/99901.html"/>
<dc:source rdf:resource="http://www.compapp.dcu.ie/~humphrys/PhD/../Publications/f.SOCO96.ps.gz"/>
<dc:language>en</dc:language>
<references rdf:resource="oai:CiteSeerPSU:18816"/>
<references rdf:resource="oai:CiteSeerPSU:194839"/>
<references rdf:resource="oai:CiteSeerPSU:163604"/>
<references rdf:resource="oai:CiteSeerPSU:87435"/>
<references rdf:resource="oai:CiteSeerPSU:142710"/>
<referencedBy rdf:resource="oai:CiteSeerPSU:8115"/>
<referencedBy rdf:resource="oai:CiteSeerPSU:230110"/>
<referencedBy rdf:resource="oai:CiteSeerPSU:69071"/>
<referencedBy rdf:resource="oai:CiteSeerPSU:32645"/>
<dc:rights>unrestricted</dc:rights>
</Paper>
</rdf:RDF>
```

A.2.3:RDF-NTRIPLES

```
<http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys>
<http://sw.deri.org/2005/07/CiteSeer/affiliation> "University of Cambridge , Computer Laboratory" .
<http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys> <http://xmlns.com/foaf/0.1/name> "Mark
Humphrys" .
<http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys> <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys> <http://sw.deri.org/2005/07/CiteSeer/authorOf>
<oai:CiteSeerPSU:99901> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/referencedBy> <oai:CiteSeerPSU:230110> .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/language> "en" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/date> "1996-06-13" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/identifier> "oai:CiteSeerPSU:99901" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:194839> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/setSpec> "CiteSeerPSUset" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/publisher> "unknown" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/contributor> "The Pennsylvania State University
CiteSeer Archives" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/referencedBy> <oai:CiteSeerPSU:32645> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:163604> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:87435> .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/description> "Reinforcement Learning (RL)
methods, in contrast tomany forms of machine learning, build up value functionsfor actions. That is, an
agent not only knows`what' it wants to do, it also knows `how much' it wantsto do it. Traditionally, the
latter are used to producethe former and are then ignored, since the agent is assumedto act alone. But the
latter numbers contain usefulinformation - they tell us how much the agent willsuffer if its action is not
executed (perhaps not much).They tell us which actions the agent can compromiseon and which it cannot. It
is clear that many interestingsystems possess multiple parallel and conflictinggoals, all demanding
attention, and none of whichcan be fully satisfied except at the expense of others.Animals are the prime
example of such systems. In[Humphrys, 1995], I introduced the W-learning algorithm,showing one method of
```

```
resolving competitionamong behaviors automatically by reference to theirRL values. The scheme has the
unusual featu..." .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/creator> "Mark Humphrys" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/authorPerson>
<http://sw.deri.org/2005/07/CiteSeer/authors/Mark_Humphrys> .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/subject> "Mark Humphrys Action selection in a
hypothetical house robot: Using those RL numbers" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/source>
<http://www.compapp.dcu.ie/~humphrys/PhD/../Publications/f.SOCO96.ps.gz> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/referencedBy> <oai:CiteSeerPSU:8115> .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/rights> "unrestricted" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/format> "ps" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/identifier>
<http://citeseer.ist.psu.edu/99901.html> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/datestamp> "1996-06-13" .
<oai:CiteSeerPSU:99901> <http://purl.org/dc/elements/1.1/title> "Action selection in a hypothetical house
robot: Using those RL numbers" .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:18816> .
<oai:CiteSeerPSU:99901> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2005/07/CiteSeer/Paper> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/referencedBy> <oai:CiteSeerPSU:69071> .
<oai:CiteSeerPSU:99901> <http://sw.deri.org/2005/07/CiteSeer/references> <oai:CiteSeerPSU:142710> .
```

B: PATENT REDBOOK XML TO RDF-NTRIPLES CONVERSION

B.1: XSLT USED

```
<?xml version="1.0"?>
<!DOCTYPE stylesheet [
<!ENTITY cr "<xsl:text>
</xsl:text>">
]>

<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns="http://sw.deri.org/2006/02/USPatent/"
  version="1.0">
<xsl:output indent="yes" />

<!-- general element conversions -->
<xsl:variable name="docNum">
     <!--<xsl:value-of select='number(/us-patent-grant/us-bibliographic-data-grant/publication-
reference/document-id/doc-number)' />-->
    <xsl:call-template name="removezeroes">
          <xsl:with-param name="text"
            select="/us-patent-grant/us-bibliographic-data-grant/publication-reference/document-id/doc-
number"/>
          <xsl:with-param name="letter"
           select=""/>
    </xsl:call-template>
</xsl:variable>

<xsl:template name="removezeroes">
  <!-- named template called by main template below -->
  <xsl:param name="text" />
  <xsl:param name="letter" />
  <xsl:variable name="int">
     <xsl:value-of select='number($text)' />
  </xsl:variable>

  <xsl:choose>
  <xsl:when test = "string-length($text) = 1">
    <xsl:choose>
    <xsl:when test = "$int = 'NaN'">
      <xsl:value-of select = "concat($letter,$text)" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select = "concat($letter,$int)" />
    </xsl:otherwise>
    </xsl:choose>
  </xsl:when>

  <xsl:when test = "$int = 'NaN'">
    <xsl:call-template name="removezeroes">
      <xsl:with-param name="text"
           select="substring($text,2)"/>
      <xsl:with-param name="letter"
           select="concat($letter,substring($text,1,1))"/>
    </xsl:call-template>
  </xsl:when>

  <!-- If no more iterations to do, add computed value to result tree. -->
  <xsl:otherwise>
   <xsl:value-of select="concat($letter,$int)"/>
  </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:variable name="uspto">
<xsl:value-of select='concat("http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=",$docNum)' />
</xsl:variable>

<xsl:variable name="docID">
<xsl:value-of select='concat("http://sw.deri.org/2006/02/USPatent/PatentID#",$docNum)' />
</xsl:variable>

<xsl:template match="text()">

</xsl:template>

<xsl:template match="/">
  <rdf:RDF>
```

```
    <xsl:apply-templates/>
  </rdf:RDF>
</xsl:template>

<xsl:template match="us-patent-grant">
  <Grant>
    <xsl:attribute name="rdf:about">
      <xsl:value-of select = '$docID' />
    </xsl:attribute>
    <usptoLink>
      <xsl:attribute name="rdf:resource">
        <xsl:value-of select = '$uspto' />
      </xsl:attribute>
    </usptoLink>
    <xsl:apply-templates/>
  </Grant>
</xsl:template>

<xsl:template match="us-bibliographic-data-grant">
  <xsl:if test = "classification-locarno">
    <internationalClassification><xsl:value-of select="classification-locarno/main-classification"
/></internationalClassification>
  </xsl:if>
  <xsl:if test = "classification-national">
    <parentUSClassification><xsl:value-of select="translate(substring(classification-national/main-
classification, 1, 3),' ','')" /></parentUSClassification>
    <mainUSClassification>
      <xsl:value-of select="translate(substring(classification-national/main-classification, 1, 3),'
','')" />
      <xsl:value-of select="'/'" />
      <xsl:value-of select="normalize-space(substring(classification-national/main-classification, 4, 3))"
/>
      <xsl:variable name = "end" select = 'substring(classification-national/main-classification, 7)' />
      <xsl:choose>
        <xsl:when test="$end = ''" />
        <xsl:when test="starts-with($end,'-')">
          <xsl:value-of select="$end" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="'.'" />
          <xsl:value-of select="$end" />
        </xsl:otherwise>
      </xsl:choose>
    </mainUSClassification>
    <xsl:for-each select="classification-national/further-classification">
      <furtherUSClassification>
        <xsl:value-of select="translate(substring(., 1, 3),' ','')" />
        <xsl:value-of select="'/'" />
        <xsl:value-of select="normalize-space(substring(., 4, 3))" />
        <xsl:variable name = "end" select = 'substring(., 7)' />
        <xsl:choose>
          <xsl:when test="$end = ''" />
          <xsl:when test="starts-with($end,'-')">
            <xsl:value-of select="$end" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="'.'" />
            <xsl:value-of select="$end" />
          </xsl:otherwise>
        </xsl:choose>
      </furtherUSClassification>
      <parentUSClassification><xsl:value-of select="normalize-space(substring(.,1,3))"
/></parentUSClassification>
    </xsl:for-each>
  </xsl:if>

  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="publication-reference">
  <xsl:if test = "document-id/doc-number">
    <number><xsl:value-of select="$docNum" /></number>
  </xsl:if>
  <xsl:if test = "document-id/country">
    <country><xsl:value-of select="document-id/country" /></country>
  </xsl:if>
  <xsl:if test= "document-id/date">
    <dc:date>
      <xsl:choose>
        <xsl:when test="string-length(document-id/date)='8'">
          <xsl:value-of select="substring(document-id/date, 1, 4)" />
          <xsl:value-of select="'-'" />
          <xsl:value-of select="substring(document-id/date, 5, 2)" />
          <xsl:value-of select="'-'" />
          <xsl:value-of select="substring(document-id/date, 7, 2)" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="document-id/date" />
        </xsl:otherwise>
      </xsl:choose>
    </dc:date>
  </xsl:if>
</xsl:template>

<xsl:template match="us-term-of-grant">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="length-of-grant">
  <term><xsl:value-of select="." /></term>
</xsl:template>

<xsl:template match="invention-title">
  <dc:title><xsl:value-of select="." /></dc:title>
</xsl:template>

<xsl:template match="references-cited">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="citation">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match = "us-field-of-classification-search">
<xsl:for-each select="classification-national/main-classification">
&cr;
  <usFOCS>
    <xsl:value-of select="translate(substring(., 1, 3),' ','')" />
    <xsl:value-of select="'/'" />
    <xsl:value-of select="normalize-space(substring(., 4, 3))" />
    <xsl:variable name = "end" select = 'substring(., 7)' />
    <xsl:choose>
      <xsl:when test="$end = ''" />
      <xsl:when test="starts-with($end,'-')">
        <xsl:value-of select="$end" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="'.'" />
        <xsl:value-of select="$end" />
      </xsl:otherwise>
    </xsl:choose>
  </usFOCS>
</xsl:for-each>
</xsl:template>

<xsl:template match="patcit">
&cr;
  <xsl:variable name = "country" select = 'document-id/country' />
  <xsl:choose>
    <xsl:when test="$country = 'US'">
      <xsl:variable name="docNumb">
        <xsl:call-template name="removezeroes">
          <xsl:with-param name="text"
            select="document-id/doc-number"/>
          <xsl:with-param name="letter"
           select=""/>
        </xsl:call-template>
      </xsl:variable>
      <citesUSPatent>
        <Grant>
          <xsl:attribute name="rdf:about">
            <xsl:value-of select = 'concat("http://sw.deri.org/2006/02/USPatent/PatentID#",$docNumb)' />
          </xsl:attribute>
          <citedByUSPatent>
            <xsl:attribute name="rdf:resource">
              <xsl:value-of select = 'concat("http://sw.deri.org/2006/02/USPatent/PatentID#",$docNum)' />
            </xsl:attribute>
          </citedByUSPatent>
        </Grant>
      </citesUSPatent>
    </xsl:when>
    <xsl:otherwise>
      <citesOther><xsl:value-of select = "concat('Patent: Country-', $country, ' Number-', document-
id/doc-number)" /></citesOther>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="parties">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="applicants">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="applicant">
&cr;
  <xsl:variable name = "fullname" select = 'concat(addressbook/first-name," ",addressbook/last-name)' />
  <applicant><xsl:value-of select="$fullname" /></applicant>
</xsl:template>

<xsl:template match="agents">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="agent">
&cr;
<xsl:choose>
  <xsl:when test="addressbook/orgname">
    <agent><xsl:value-of select="addressbook/orgname" /></agent>
  </xsl:when>
  <xsl:when test="addressbook/first-name">
    <xsl:variable name = "fullname" select = 'concat(addressbook/first-name," ",addressbook/last-name)' />
    <agent><xsl:value-of select="$fullname" /></agent>
  </xsl:when>
  <xsl:otherwise />
</xsl:choose>
</xsl:template>

<xsl:template match="assignees">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="assignee">
&cr;
<xsl:choose>
  <xsl:when test="addressbook/orgname">
    <assignee><xsl:value-of select="addressbook/orgname" /></assignee>
  </xsl:when>
  <xsl:when test="addressbook/first-name">
    <xsl:variable name = "fullname" select = 'concat(addressbook/first-name," ",addressbook/last-name)' />
    <assignee><xsl:value-of select="$fullname" /></assignee>
  </xsl:when>
  <xsl:otherwise />
</xsl:choose>
</xsl:template>

<xsl:template match="examiners">
&cr;
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="primary-examiner">
&cr;
  <xsl:variable name = "fullname" select = 'concat(first-name," ",last-name)' />
  <primaryExaminer><xsl:value-of select="$fullname" /></primaryExaminer>
</xsl:template>

<xsl:template match="assistant-examiner">
&cr;
  <xsl:variable name = "fullname" select = 'concat(first-name," ",last-name)' />
  <assistantExaminer><xsl:value-of select="$fullname" /></assistantExaminer>
</xsl:template>

<xsl:template match ="abstract">
&cr;
  <xsl:for-each select="p">
    <dc:description>
      <xsl:value-of select='concat("{", substring(@id,3) , "}" , .)' />
    </dc:description>
  </xsl:for-each>
</xsl:template>
</xsl:transform>
```

B.2:CONVERSION PATH OF A SINGLE PATENT ENTRY

B.2.1:REDBOOK(XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<us-patent-grant  lang="EN"  dtd-version="v40  2004-12-02"  file="USD0500576-20050111.XML"  status="SAMPLE-
DATA" id="us-patent-grant" country="US" date-produced="20041228" date-publ="20050111">
<us-bibliographic-data-grant>
<publication-reference>
<document-id>
<country>US</country>
<doc-number>D0500576</doc-number>
<kind>S1</kind>
<date>20050111</date>
</document-id>
</publication-reference>
<application-reference appl-type="design">
<document-id>
<country>US</country>
<doc-number>29178812</doc-number>
<date>20030331</date>
</document-id>
</application-reference>
<us-application-series-code>29</us-application-series-code>
<us-term-of-grant>
<length-of-grant>14</length-of-grant>
</us-term-of-grant>
<classification-locarno>
<edition>7</edition>
<main-classification>0101</main-classification>
</classification-locarno>
<classification-national>
<country>US</country>
<main-classification>D 1199</main-classification>
<further-classification> D1125</further-classification>
<further-classification>426104</further-classification>
<further-classification>D21386</further-classification>
</classification-national>
<invention-title id="d0e53">Food blend product</invention-title>
<references-cited>
<citation>
<patcit num="00001">
<document-id>
<country>US</country>
<doc-number>450378</doc-number>
<kind>A</kind>
<name>Robinson</name>
<date>18910400</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>451461</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00002">
<document-id>
<country>US</country>
<doc-number>D174778</doc-number>
<kind>S</kind>
<name>Smith</name>
<date>19550500</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>D                           6601</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00003">
<document-id>
<country>US</country>
<doc-number>3589914</doc-number>
<kind>A</kind>
<name>Cooper et al.</name>
<date>19710600</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>426104</main-
classification></classification-national>
</citation>
```

```
<citation>
<patcit num="00004">
<document-id>
<country>US</country>
<doc-number>4192899</doc-number>
<kind>A</kind>
<name>Roth</name>
<date>19800300</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>426513</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00005">
<document-id>
<country>US</country>
<doc-number>D296377</doc-number>
<kind>S</kind>
<name>Skiver et al.</name>
<date>19880600</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>D                           1199</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00006">
<document-id>
<country>US</country>
<doc-number>5447584</doc-number>
<kind>A</kind>
<name>Shakespeare et al.</name>
<date>19950900</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>156                           63</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00007">
<document-id>
<country>US</country>
<doc-number>5458433</doc-number>
<kind>A</kind>
<name>Stastny</name>
<date>19951000</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>4034081</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00008">
<document-id>
<country>US</country>
<doc-number>D393345</doc-number>
<kind>S</kind>
<name>Clegg et al.</name>
<date>19980400</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>D                           2869</main-
classification></classification-national>
</citation>
<citation>
<patcit num="00009">
<document-id>
<country>US</country>
<doc-number>D402227</doc-number>
<kind>S</kind>
<name>Granger et al.</name>
<date>19981200</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>D11131</main-
classification></classification-national>
</citation>
```

```
<citation>
<patcit num="00010">
<document-id>
<country>US</country>
<doc-number>D409356</doc-number>
<kind>S</kind>
<name>Vodhanel, Jr.</name>
<date>19990500</date>
</document-id>
</patcit>
<category>cited by examiner</category>
<classification-national><country>US</country><main-classification>D                              1199</main-
classification></classification-national>
</citation>
</references-cited>
<number-of-claims>1</number-of-claims>
<us-exemplary-claim>1</us-exemplary-claim>
<field-of-search>
<classification-national>
<country>US</country>
<main-classification>D 1199</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>D 1125-127</main-classification>
<additional-info>unstructured</additional-info>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>D 1106</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>426104</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>426513</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>426 92</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>426558-560</main-classification>
<additional-info>unstructured</additional-info>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>D21385-389</main-classification>
<additional-info>unstructured</additional-info>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>D21484</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>446 85</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>273288</main-classification>
</classification-national>
<classification-national>
<country>US</country>
<main-classification>D11131</main-classification>
</classification-national>
</field-of-search>
<figures>
<number-of-drawing-sheets>2</number-of-drawing-sheets>
<number-of-figures>9</number-of-figures>
</figures>
<parties>
<applicants>
<applicant sequence="001" app-type="applicant-inventor" designation="us-only">
<addressbook>
<last-name>Kraus</last-name>
<first-name>Gerald L.</first-name>
<address>
<city>Plymouth</city>
<state>WI</state>
<country>US</country>
```

```
</address>
</addressbook>
<nationality>
<country>omitted</country>
</nationality>
<residence>
<country>US</country>
</residence>
</applicant>
</applicants>
<agents>
<agent sequence="01" rep-type="attorney">
<addressbook>
<orgname>Reinhart Boerner Van Deuren S.E.</orgname>
<address>
<country>unknown</country>
</address>
</addressbook>
</agent>
</agents>
</parties>
<assignees>
<assignee>
<addressbook>
<orgname>Sargento Foods, Inc.</orgname>
<role>02</role>
<address>
<city>Plymouth</city>
<state>WI</state>
<country>US</country>
</address>
</addressbook>
</assignee>
</assignees>
<examiners>
<primary-examiner>
<last-name>Burgess</last-name>
<first-name>Pamela</first-name>
<department>2911</department>
</primary-examiner>
</examiners>
</us-bibliographic-data-grant>
</us-patent-grant>
```

B.2.2:RDF-XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="http://sw.deri.org/2006/02/USPatent/">
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D500576">
<usptoLink rdf:resource="http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=D500576"/>
<internationalClassification>0101</internationalClassification>
<parentUSClassification>D1</parentUSClassification>
<mainUSClassification>D1/199</mainUSClassification>
<furtherUSClassification>D1/125</furtherUSClassification>
<parentUSClassification>D1</parentUSClassification>
<furtherUSClassification>426/104</furtherUSClassification>
<parentUSClassification>426</parentUSClassification>
<furtherUSClassification>D21/386</furtherUSClassification>
<parentUSClassification>D21</parentUSClassification>
<number>D500576</number>
<country>US</country>
<dc:date>2005-01-11</dc:date>
<term>14</term>
<dc:title>Food blend product</dc:title>


<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#450378">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D174778">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#3589914">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
```

```
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#4192899">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D296377">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#5447584">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#5458433">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D393345">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D402227">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<citesUSPatent>
<Grant rdf:about="http://sw.deri.org/2006/02/USPatent/PatentID#D409356">
<citedByUSPatent rdf:resource="http://sw.deri.org/2006/02/USPatent/PatentID#D500576"/>
</Grant>
</citesUSPatent>

<applicant>Gerald L. Kraus</applicant>

<agent>Reinhart Boerner Van Deuren S.E.</agent>

<assignee>Sargento Foods, Inc.</assignee>

<primaryExaminer>Pamela Burgess</primaryExaminer>
</Grant>
</rdf:RDF>
```

B.2.3:RDF-NTRIPLES

```
<http://sw.deri.org/2006/02/USPatent/PatentID#D409356>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D409356>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D393345>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D393345>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#3589914>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#3589914>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#450378>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#450378>    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D296377>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D296377>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D174778>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
```

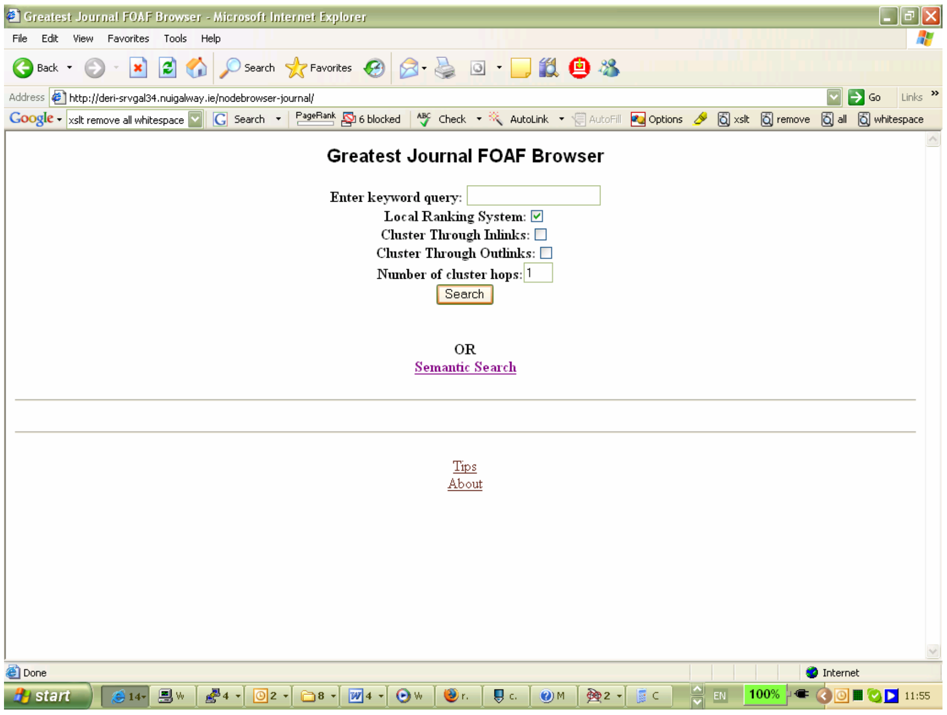
```
<http://sw.deri.org/2006/02/USPatent/PatentID#D174778>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#5447584>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#5447584>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D402227>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D402227>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#5458433>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#5458433>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#4192899>
<http://sw.deri.org/2006/02/USPatent/citedByUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#4192899>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>        <http://sw.deri.org/2006/02/USPatent/number>
"D500576" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>         <http://sw.deri.org/2006/02/USPatent/agent>
"Reinhart Boerner Van Deuren S.E." .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/parentUSClassification> "426" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#5458433> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/parentUSClassification> "D21" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D296377> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/internationalClassification> "0101" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://purl.org/dc/elements/1.1/date> "2005-01-11"
.
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#450378> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D174778> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D402227> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/furtherUSClassification> "D21/386" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/term> "14" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D393345> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>      <http://sw.deri.org/2006/02/USPatent/assignee>
"Sargento Foods, Inc." .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://sw.deri.org/2006/02/USPatent/Grant> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://purl.org/dc/elements/1.1/title> "Food blend
product" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#5447584> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>     <http://sw.deri.org/2006/02/USPatent/usptoLink>
<http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=D500576> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/furtherUSClassification> "426/104" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/furtherUSClassification> "D1/125" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>     <http://sw.deri.org/2006/02/USPatent/applicant>
"Gerald L. Kraus" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/primaryExaminer> "Pamela Burgess" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>  <http://sw.deri.org/2006/02/USPatent/country>  "US"
.
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/mainUSClassification> "D1/199" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#4192899> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#3589914> .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576>
<http://sw.deri.org/2006/02/USPatent/parentUSClassification> "D1" .
<http://sw.deri.org/2006/02/USPatent/PatentID#D500576> <http://sw.deri.org/2006/02/USPatent/citesUSPatent>
<http://sw.deri.org/2006/02/USPatent/PatentID#D409356> .
```

C: EXAMPLE CONFIG.XML - CITESEER

```
<config>
<predicate>
  <uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</uri>
  <label>is</label>
  <order>1</order>
  <internal>0</internal>
</predicate>

<predicate>
  <uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#seeAlso</uri>
  <label>Has foaf page</label>
  <order>2</order>
  <internal>0</internal>
</predicate>

<predicate>
  <uri>http://purl.org/dc/elements/1.1/title</uri>
  <label>Title</label>
  <order>3</order>
</predicate>

<predicate>
  <uri>http://purl.org/dc/elements/1.1/description</uri>
  <label>Description</label>
  <order>4</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/nick</uri>
  <label>Has Username</label>
  <order>5</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/dateOfBirth</uri>
  <label>Was born</label>
  <order>6</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/mbox_sha1sum</uri>
  <label>Has Email sha1sum</label>
  <order>7</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/page</uri>
  <label>Has Page</label>
  <order>8</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/aimChatID</uri>
  <label>Has aimChatID</label>
  <order>9</order>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/homepage</uri>
  <label>Has Homepage</label>
  <order>10</order>
  <internal>0</internal>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/weblog</uri>
  <label>Has Weblog</label>
  <order>11</order>
  <internal>0</internal>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/interest</uri>
  <label>Has Interest</label>
  <order>12</order>
  <internal>1</internal>
</predicate>

<predicate>
  <uri>http://xmlns.com/foaf/0.1/knows</uri>
  <label>knows</label>
  <order>13</order>
  <internal>1</internal>
</predicate>

<predicate>
  <uri>http://sw.deri.org/2005/05/ranking#dynamicRankScore</uri>
  <label>Dynamic Rank Score</label>
  <order>14</order>
</predicate>

<predicate>
  <uri>http://sw.deri.org/2005/05/ranking#dynamicClusteredState</uri>
  <label>Clustered state (direction and hops if not false)</label>
  <order>15</order>
</predicate>

<subject>
  <type>http://xmlns.com/foaf/0.1/Person</type>
  <typelabel>Person</typelabel>
  <divclass>person</divclass>
</subject>

<subject>
  <type>http://xmlns.com/foaf/0.1/Document</type>
  <typelabel>Document</typelabel>
  <divclass>Document</divclass>
</subject>
</config>
```

D: SCREENSHOTS

D.1:SCREENSHOT OF INITIAL SCREEN


Greatest Journal FOAF Browser - Microsoft Internet Explorer
http://deri-srvgal34.nuigalway.ie/nodebrowser-journal/

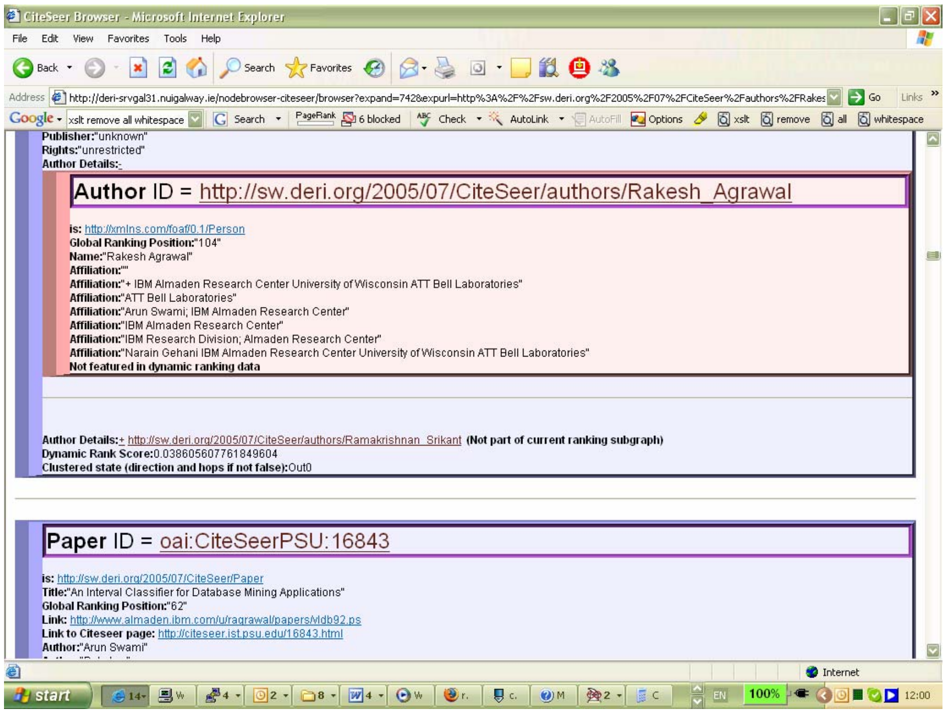
Greatest Journal FOAF Browser

Enter keyword query:
Local Ranking System: ✓
Cluster Through Inlinks:
Cluster Through Outlinks:
Number of cluster hops: 1
Search

OR
Semantic Search

Tips
About

D.2:SCREENSHOT OF TABBED BROWSING


CiteSeer Browser - Microsoft Internet Explorer
http://deri-srvgal31.nuigalway.ie/nodebrowser-citeseer/browser?expand=742&expurl=http%3A%2F%2Fsw.deri.org%2F2005%2F07%2FCiteSeer%2Fauthors%2FRakes

Publisher:"unknown"
Rights:"unrestricted"
Author Details:-

Author ID = http://sw.deri.org/2005/07/CiteSeer/authors/Rakesh_Agrawal

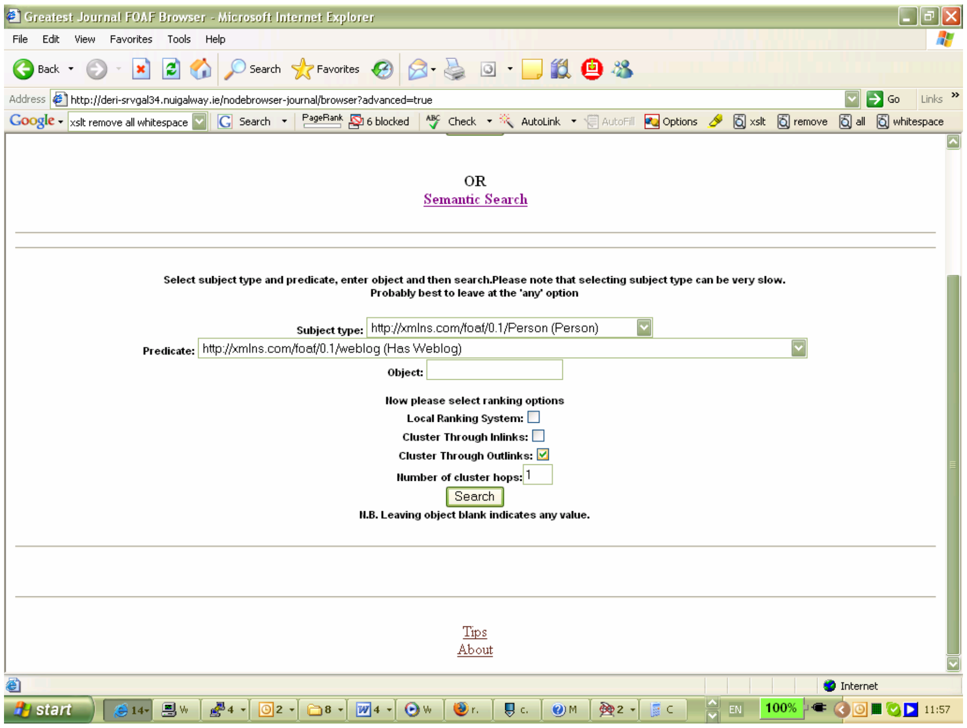
is: http://xmlns.com/foaf/0.1/Person
Global Ranking Position:"104"
Name:"Rakesh Agrawal"
Affiliation:""
Affiliation:"+ IBM Almaden Research Center University of Wisconsin ATT Bell Laboratories"
Affiliation:"ATT Bell Laboratories"
Affiliation:"Arun Swami; IBM Almaden Research Center"
Affiliation:"IBM Almaden Research Center"
Affiliation:"IBM Research Division; Almaden Research Center"
Affiliation:"Narain Gehani IBM Almaden Research Center University of Wisconsin ATT Bell Laboratories"
Not featured in dynamic ranking data

Author Details:+ http://sw.deri.org/2005/07/CiteSeer/authors/Ramakrishnan_Srikant **(Not part of current ranking subgraph)**
Dynamic Rank Score:0.038605607761849604
Clustered state (direction and hops if not false): Out0

Paper ID = oai:CiteSeerPSU:16843

is: http://sw.deri.org/2005/07/CiteSeer/Paper
Title:"An Interval Classifier for Database Mining Applications"
Global Ranking Position:"62"
Link: http://www.almaden.ibm.com/u/ragrawal/papers/vldb92.ps
Link to Citeseer page: http://citeseer.ist.psu.edu/16843.html
Author:"Arun Swami"

D.3:SCREENSHOT OF ADVANCED BROWSING FORM


Greatest Journal FOAF Browser - Microsoft Internet Explorer
Address http://deri-srvgal34.nuigalway.ie/nodebrowser-journal/browser?advanced=true
Search
Tips
About

OR
Semantic Search

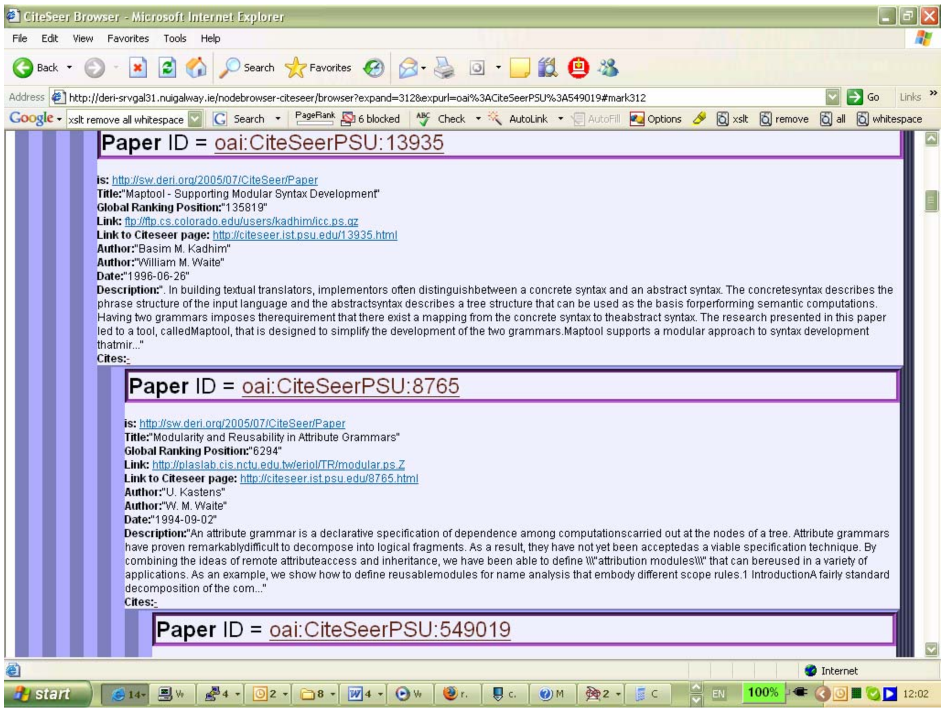
Select subject type and predicate, enter object and then search.Please note that selecting subject type can be very slow.
Probably best to leave at the 'any' option

Subject type: http://xmlns.com/foaf/0.1/Person (Person)
Predicate: http://xmlns.com/foaf/0.1/weblog (Has Weblog)
Object:

Now please select ranking options
Local Ranking System: ☐
Cluster Through Inlinks: ☐
Cluster Through Outlinks: ✓
Number of cluster hops: 1

N.B. Leaving object blank indicates any value.

D.4:SCREENSHOT OF TREE BROWSING


CiteSeer Browser - Microsoft Internet Explorer
Address http://deri-srvgal31.nuigalway.ie/nodebrowser-citeseer/browser?expand=312&expurl=oai%3ACiteSeerPSU%3A549019#mark312

Paper ID = oai:CiteSeerPSU:13935

is: http://sw.deri.org/2005/07/CiteSeer/Paper
Title:"Maptool - Supporting Modular Syntax Development"
Global Ranking Position:"135819"
Link: ftp://ftp.cs.colorado.edu/users/kadhim/icc.ps.gz
Link to Citeseer page: http://citeseer.ist.psu.edu/13935.html
Author:"Basim M. Kadhim"
Author:"William M. Waite"
Date:"1996-06-26"
Description:". In building textual translators, implementors often distinguishbetween a concrete syntax and an abstract syntax. The concretesyntax describes the phrase structure of the input language and the abstractsyntax describes a tree structure that can be used as the basis forperforming semantic computations. Having two grammars imposes therequirement that there exist a mapping from the concrete syntax to theabstract syntax. The research presented in this paper led to a tool, calledMaptool, that is designed to simplify the development of the two grammars.Maptool supports a modular approach to syntax development thatmir..."
Cites:-

Paper ID = oai:CiteSeerPSU:8765

is: http://sw.deri.org/2005/07/CiteSeer/Paper
Title:"Modularity and Reusability in Attribute Grammars"
Global Ranking Position:"6294"
Link: http://plaslab.cis.nctu.edu.tw/eriol/TR/modular.ps.Z
Link to Citeseer page: http://citeseer.ist.psu.edu/8765.html
Author:"U. Kastens"
Author:"W. M. Waite"
Date:"1994-09-02"
Description:"An attribute grammar is a declarative specification of dependence among computationscarried out at the nodes of a tree. Attribute grammars have proven remarkablydifficult to decompose into logical fragments. As a result, they have not yet been acceptedas a viable specification technique. By combining the ideas of remote attributeaccess and inheritance, we have been able to define \\\"attribution modules\\\" that can bereused in a variety of applications. As an example, we show how to define reusablemodules for name analysis that embody different scope rules.1 IntroductionA fairly standard decomposition of the com..."
Cites:-

Paper ID = oai:CiteSeerPSU:549019