SmartHR: A Resume Query and Management System Based on Semantic Web

Yeqing Ke, Zhirou Ma, Haijiang Wu, Jie Liu, Hua Zhong, Jun Wei
Institute of Software, Chinese Academy of Sciences
{keyeqing12, mazhirou, wuhaijiang12, ljie, zhongh, wj}@otcaix.iscas.ac.cn

ABSTRACT

Organizations are always confronted with the challenge of efficiently finding out suitable candidates from massive resumes. Traditional human resource management based on the information management system usually adopts SQL queries or keywords search, which cannot capture the implicit information, while the manual work is always time-consuming. To fill this gap, this paper presents *SmartHR*, a resume query and management system based on semantic web. Benefiting from knowledge base, it can understand users' intentions more intelligently and search for suitable candidates more accurately. In this paper, we propose two key technical difficulties which SmartHR meets, including the complexity of knowledge base construction and the time-consuming semantic search, and then give appropriate solutions respectively. Four channels are adopted to construct knowledge base, which are well illustrated. Furthermore, a variety of performance optimizations are employed and the effectiveness is evaluated on real datasets of up to millions of triples and the results show a great improvement. As a representative application in semantic web, our practice in SmartHR provides useful experience and conclusions for developers.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

Semantic Web, RDF, SPARQL, Performance Optimization

CrowdSoft'14, November 17, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3224-8/14/11...$15.00
http://dx.doi.org/10.1145/2666539.2666573

1. INTRODUCTION

With the rapid development of globalization, unprecedented fierce competition makes organizations have a greater awareness of talents' importance. Unfortunately, how to find the suitable candidates from numerous resumes accurately and efficiently poses challenges to human resources management. The resumes are always semi-structured data consisting of complex information. Traditional process, which relies heavily on HR, is manual intensive and time-consuming. To alleviate this pressure, a variety of techniques, such as information retrieve, have been adopted and gained a reasonable degree of maturity. In general, these techniques are based almost purely on the occurrence of keyword and tend to get the exactly-matched results. However, they can hardly meet the requirements for many situations. For example, one may want to find out the implicit information through query "who once worked in poor areas?", while the resumes only contain the persons' work locations such as provinces, cities and counties, rather than the "poor area" keyword.

To address this problem, we present *SmartHR*, a resume query and management system based on semantic web. Benefiting from knowledge base, it can understand users' intentions more intelligently and search for suitable candidates more accurately. The semantic web [8] has been widely applied due to its great potential for providing a common framework that allows data to be shared and reused across applications and enterprises. In SmartHR, semantic annotations for various heterogeneous resources are represented as RDF [2] data model and the query language SPARQL is leveraged to figure out triples. Benefiting from these advantages, SmartHR provides not only the extensible mechanism for data source management, but also the interfaces for users to allow for sophisticated semantic search on resumes. It facilitates users, who are unfamiliar with SPARQL or underlying ontology, to explore the RDF annotated data.

This paper presents two essential techniques for building up SmartHR, the typical semantic web application. First we construct knowledge base by four channels, including mapping from relational database to RDF data store, knowledge base maintenance, semantic extension and semantic parsing. Second, we make a variety of performance optimizations of the semantic search engine from the aspects of SPARQL query rewriting, hybrid storage strategy for RDF data and in-memory cache adoption. The experimental results show the effectiveness of these optimizations. Our main contributions are as follows.

- We build a powerful ontology to characterize human resource management and make a real-world case s-

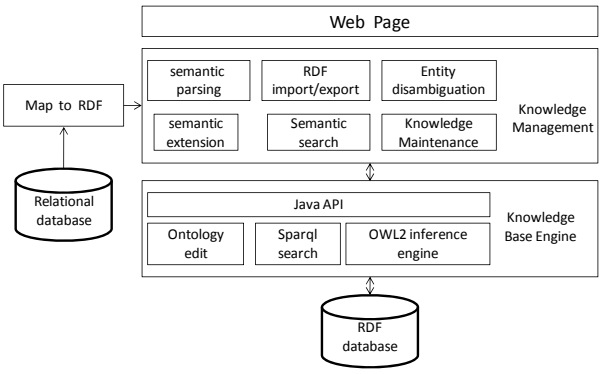

Figure 1: The System Architecture of SmartHR.

tudy for semantic web related techniques, offering useful experience and conclusions for developers.

- We propose four channels to construct the knowledge base. The process of construction is always complex and confronted with many problems, such as ambiguity which should be eliminated.
- We make various performance optimizations for semantic search. The results show that they can improves the query performance.

The remainder of this paper is organized as follows. In Section 2, we make an overview of SmartHR's architecture design. Section 3 presents its technical details of knowledge base construction. Performance optimizations are demonstrated in Section 4. Section 5 demonstrates the effectiveness from experiments and we introduce the related work in section 6. Conclusions and future works are summarized in Section 7.

2. OVERVIEW OF SMARTHR

This section makes an overview of SmartHR's architecture design. As demonstrated in Figure 1, it mainly consists of four components: the transformation component which maps relational data into RDF storage, the knowledge base engine component, the knowledge management and web page components. Moreover, these components in SmartHR are loosely coupled, that is, they can even be adopted separately by other applications.

The left part in Figure 1 is the transformer, which takes semi-structured relational data extracted from resumes as its input. In our practice, semi-structured resumes are usually processed as tuples (such as *name*, *age* and the corresponding *description* for a person) to be stored in relational databases. This design makes SmartHR easy to integrate with other applications, such as the online e-recruitment systems. The web component is a present layer which provides friendly interfaces for users to query data and get the results. Components about the knowledge base are the central of SmartHR. It relies on a dynamic knowledge management strategy, and contains a variety of tools to meet users different functional requirements. In knowledge base engine, Java API plays as an intermediate layer between the underlying search engines and the upper knowledge management layer.

It can be seen that, SmartHR gains a flexible and extensible architecture. In the following sections, we will mainly concentrate on two essential categories, the knowledge

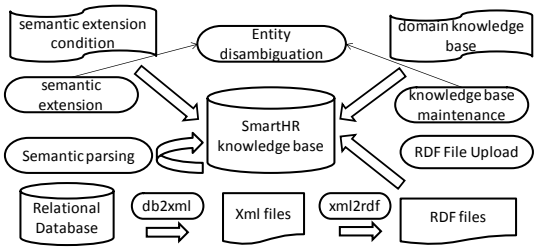

Figure 2: Knowledge Base Construction.

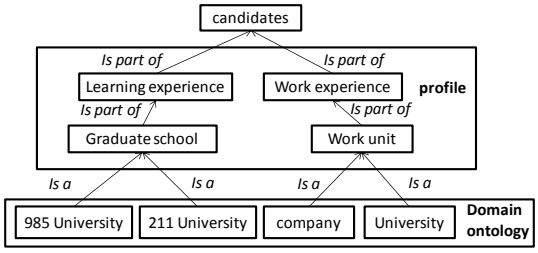

Figure 3: A Brief Scheme of Ontology.

base construction and performance optimization, to reveal the technical details and challenges in semantic web application.

3. KNOWLEDGE BASE CONSTRUCTION

Knowledge base is the foundation of semantic search. Our system constructs knowledge base through four channels shown in Figure 2. Fisrt of all, we get data from existing information in relational databases and transform it into xml files. And then the xml files are mapped to RDF files so that we can import the RDF files into our RDF data store. Besides, we can manage our knowledge base through domain knowledge maintenance, semantic extension and semantic parsing.

3.1 Relational Database to RDF Mapping

3.1.1 Ontology Design

By taking into account all considerations, ontology for SmartHR has been designed. A brief scheme of the ontology is shown in Figure 3. The main concepts are 'candidate', 'profile' and some specified domain concepts which are named classes in SmartHR. All the main concepts and relationships relevant to them are detailed in the following sections.

Concepts. In our system, concepts are always classes which are a set of individuals, including 'candidate', 'profile' and some specific domain concepts. 'candidate' represents a set of job candidates. The properties defined for the concept are: name, date of birth, gender, country, province and city. The candidates have to specify their abilities in order to create their own profile. 'profile' is central in the application since it determines whether an offer is suitable for a candidate or not. It is mainly composed by two parts: learning experience and work experience. Learning experience includes information about candidates' graduate school, highest school, major and so on. Work experience includes information about candidates' work units, positions and so on.

This ontology also contains many specific domain concepts, such as University, 985 University, Enterprise and so on, which help us find out suitable candidates.

Relations. Two types of inter conceptual relations are used in the system, namely, 'Part-of' and 'IS-A'. A 'Part-of' relation means that one concept is a part of another one. In the SmartHR, the relation of 'Part-of' is divided into two types, namely, object property and data property. Object property is the relation between individuals and data property is the relation between individual and literals. The 'IS-A' relation is used for creating categorical structures. In our system, it is implemented as the relation between classes and the relation between individuals and classes. For example, the 985 University is subclass of the university class and meanwhile the Peking University is an individual of 985 University.

Individuals. A set of individuals compose a class. For example, the class of candidate is intended to describe a set of candidates, such as Mary, John and so on.

3.1.2 Mapping RDBMS to RDF Implementation

The candidates' resume information stored in relational database is shown in the following tables (Table 1, Table 2 and Table 3). According to the ontology designed above, two main steps are adopted to automatically map the original relational database to RDF data store.

Table 1: Basis

uid	name	age
1	Mary	34

Table 2: Learning Experience

id	uid	school	start
2	1	Peking University	1996

Table 3: Work Experience

id	uid	location and position
3	1	Beijing, manager

Step1: First, it extracts useful information such as candidate's name, age, learning experience, work experience and so on from the original relational database and transforms it to semi-structured data stored as XML files using Jdom2. Then it preprocesses unstructured words such as "Location and position" by word segmentation and generate xml files shown in Figure 4. Step2: According to classes, individuals and properties in the ontology designed above, it converts xml files to RDF files by a recursive algorithm shown in Algorithm 1 and Algorithm 2. If the element in the xml file has children, an object property and an individual are created and then its children are traversed recursively. Otherwise, a data property is created. It is implemented by Jena API and the output RDF file shown in Figure 4.

3.2 Domain Knowledge Base Maintenance

The knowledge base is dynamically evolving, so the SmartHR allows to edit classes, individuals and properties. When a new class is created and a set of new individuals need to be imported, ambiguity between new individuals and the individuals of the new class's super classes should be eliminated. The process we maintain the specific domain knowledge base is demonstrated in detail as Figure 5. If the new individual is matched with the individual of the new class's super classes', the existing individual will be just added to the new class instead of being created. For example, there is a class named university, which has an individual named Tsinghua University in SmartHR. When a subclass of university named 985 university is created and a list of 985 university individuals which also contain Tsinghua University needs to be added, maybe the individual has existed in 985 university class's super class which is university class by matching algorithm. In this case, the individual, Tsinghua University, is just added to 985 university class created so that ambiguity is avoided.

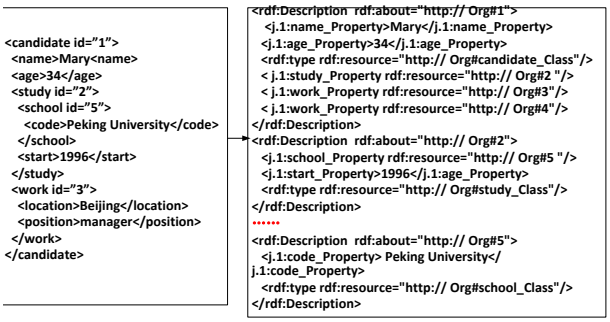

Figure 4: Transformation from xml to RDF.

Algorithm 1 XML to RDF Transformation

Input:
The XML file of the candidate's resume
Output:
The RDF file of the candidate's resume
Reading the xml file and getting the root element
Creating a new rdf file
if there is no candidate class **then**
Creating a candidate class
end if
Creating an individual of the candidate class
Traversing depth-firstly the root element of the xml and writing the rdf file. (detailed in Algorithm2.)
return the RDF file

3.3 Semantic Extension

Sometimes our individuals of a concept may be incomplete. It is necessary to extend a specific concept effectively by binding it with other individuals' property. Thus, new individuals are created if it is only a data property , otherwise individuals which the object property links to are obtained. Then they are added to the mapping class if they don't exist in the specific concept. The matching process is called entity disambiguation shown in Figure 6. For example, the University class s need to be extended. First, we bind the

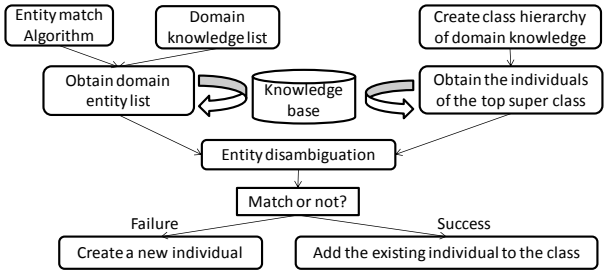

Figure 5: Domain Knowledge Base Maintenance.

Algorithm 2 Traversing depth-firstly the element in the xml and writing the rdf.

Input:
Element e; Individual $parentInd$;
for each child $elchild$ of e **do**
  if $elchild$ has no children **then**
    Creating a statement where Element e is a data property of $parentInd$
  else
    if there is no class named e's name **then**
      Creating a class named e's name.
    end if
    Creating an new individual $eind$ for e as an individual of the class named e's name.
    Creating a statement where $eind$ is a object property of $parentInd$
    Traversing depth-firstly the $elchild$.
  end if
end for

candidates' school_property p with s. If school_property is a data property, new individuals will be created which are a set of candidates' school. Second, we add the candidates' school individuals to s if the school does't exist in s.

3.4 Semantic Parsing

It is common that many elements in xml files are automatically mapped to dataproperty in RDF data store since they have no children. As we know, the node which data property links to is literals which are just values. To search candidates semantically, data property need to be parsed and converted to the object property. The process includes creating new class, adding individuals to the new class and adding statement <s,p,o> to our knowledge base where s is the individual which links to data property, p is the new object property, o is the new individual. For example, the candidate Mary's location property is a data property because it has no children in the xml file. The system can convert it to a object property p. First it creates a class named location_class and an individual named Beijing. Second, it adds the new individual into the new class. Third, a new statement <Mary, p, Beijing> is added to the knowledge base.

4. PERFORMANCE OPTIMIZATION

4.1 SPARQL Query Optimization

Query optimization is the first stage of optimization and it occurs automatically at the end of query parsing. The aim of query optimization is to reorder the triple patterns within each graph pattern to do several things: evaluate the most selective triple patterns first, evaluate FILTER at the earliest possible point and minimize the complexity of joins within graph patterns and across graph patterns. According to these rules, we rewrite the sparql queries and the time of searching doesn't reach our goals.

4.2 Storage Optimization

Vertical table stored in the storage is the most direct solution which stores RDF triples directly in a three-column table consisted of three columns including subject, predicate and object. Since all of the RDF data is stored in the same

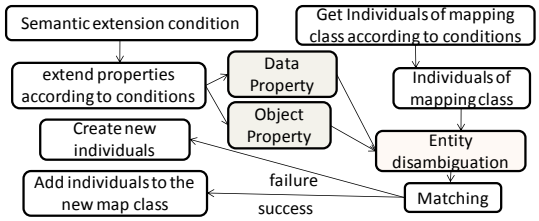

Figure 6: Semantic Extension.

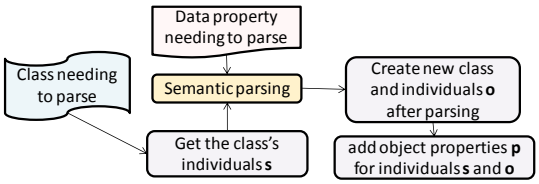

Figure 7: Semantic Parsing.

```
SELECT DISTINCT ?x ?info
 WHERE
{
        ?x p.1:Foundation_Property ?info.
        ?x rdf:type  p.1:985UniversityCandidate_Class.
}
```

Figure 8: The optimized SPARQL of 985 University.

data table, resulting in a big table with a large amount of triples and the lower query response performance. Besides, triples with the same subject and different predicates are stored in different rows so that when the SPARQL query is translated into SQL queries, there are a lot of join operations which lead to low query performance. For example, in the 985 University case, we can see a lot of join operations when we want to retrieve the candidate's name, age, sex and so on. When the table is too large, the increasing number of join operation will lead to a significant increase in query time.

4.2.1 Minimize the Join Operation

We preprocess the candidates' basic information and store them in advance as the candidates' object properties. We also make classification of the candidates in advance. And then rewrite the 985 University sparql as shown in Figure 8 in order to minimize the join operations. We can see the sparql become much more brief and when it is translated into the SQL, the join operations reduce to once. While the number of triples reaches 400000, only one join operation will lead to the poor query performance. So we adopt the optimization solution below further.

4.2.2 Hybrid Storage of Vertical Property Tables

Property tables are the n-column tables designed to store RDF data. The same subject and corresponding property values are stored in each row. However, a multi-valued problem exists when it is directly applied in SmartHR. For example, a candidate' types are multi-valued since the candidate can belong to the class of who have graduated from

985 University, the class of who have graduated from 211 University and the class of who ever worked in enterprises. In this situation, the subject of candidate has more than one object value for the property of type and then each distinct value is listed in a successive row in the table for that property. It is vertically partitioned in the multi-valued property shown in Table 4. The SQL translated from the SPARQL Query above is shown below, and we can see it has no join operation.

Select Candidate.ID, Candidate.Foundation_Property
From Candidate
Where Candidate.University_type=" 985UniversityCandidate_Class"

Table 4: Candidates

ID	Basic_Property	Type
Id1	Info1	985UniversityCandidate_Class
Id1	Info1	EntepriseCandidate_Class
Id1	Info1	211UniversityCandidate_Class
Id2	Info1	BasicLevelCandidate_Class
Id2	Info2	211UniversityCandidate_Class

4.3 In-memory Cache Optimization

We also propose an approach of in-memory cache optimization. We know memory becomes more and more cheap. When we start the server, results of queries will be loaded in advance and the index for them will be built in memory cache. Once users send a request for search, the server will search the query in memory first. If the query is hit, the results will be sent to the user immediately, otherwise the application server will send the query request to the database server and the server process the request and retrieve the results to users. The optimized cache provides better query performance by reusing previously executed results and, when possible, avoids sending new queries to the data source.

5. EXPERIMENTS

As mentioned in Section 1, performance is a significant problem in the semantic analysis for human resource management where a large number of resources need to be process. Some factors, such as the number of triples in tables and the number of query results affect the system performance. We conduct a series of experiments to study the performance of SmartHR and understand which factors seriously affect the system performance. The total number of candidates is 12000. And total numbers of quads stored in MySQL is 400,000. Our experiment environment is list in Table 5.

Table 5: Experiment Environment

CPU	Intel Core i5-3470
Memory	100GB
Tomcat Server version	7.0.47
MySQL Server version	5.5.28

5.1 Query Example

When a user enters words into the search box such as "who is graduate from 985 University", the SmartHR will segment these words and extract the keywords "985 University" to the application server. The server will translate them into the SPARQL Language Query in the Figure 11. Then the SPARQL search engine will process it and retrieve the results.

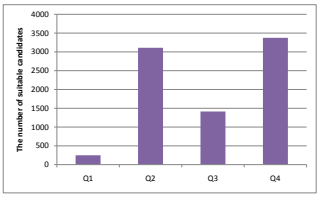

Figure 9: Suitable Candidates Number.

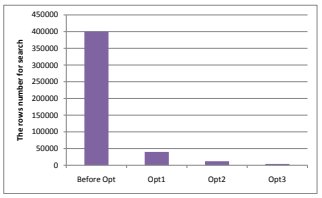

Figure 10: The Rows Number for Search.

```
SELECT DISTINCT ?name ?birthday ?sex ?position ? school ?degree
WHERE
{
    ?x p.1:name_Property ?name.
    ?x p.1:birthday_Property ?birthday.
    ?x p.1:sex_Property ?s.
    ?s rdfs:label ?sex.
    ?x p.1:position_Property ?position.
    ?x p.1:study_Property ?y.
    ?y p.1:school_Property ?m.
    ?m rdf:type  p.1:985University_Class.
    ?m rdfs:label ?school.
    ?y p.1:degree_Property ?degree.
    FILTER(regex(str(?degree),"GB68644")).
}
```

Figure 11: The SPARQL of 985 University.

5.2 Experiment Results and Analysis

We can see from Table 6, Figure 10 and Figure 9 that as the number of suitable candidates meeting the conditions and the number of rows where each query searches increase, the time of search increases greatly so that users can't tolerate. After three performance optimizations introduced above, the queries' number for each search decreases, which shown in Table 8 and Figure 12. Further, the cost time of search reduces to within 1s. Query time comparison is shown in Table 7 and Figure 13. Q1-Q3 searches for candidates who have studied in 985 universities, who have studied in military universities, who have worked in enterprises and who have worked in poor areas. Opt1 represents SPARQL query optimization. Opt2 means storage optimization after Opt2. Opt3 stands for In-memory cache optimization after opt2. Besides, we also implement combination query in memory cache, whose performance after three optimizations has significantly improved shown in Table 9 and Figure 14.

6. RELATED WORK

RDF data is a collection of statements, called triples, of the form $< s, p, o >$ where s is called subject, p is called predicate, and o is called object, and each triple states the relation between a subject and an object. Such a collection of triples can be viewed as a directed graph, in which nodes represent subjects and objects, and edges represent predicates connecting from subject nodes to object nodes. Most

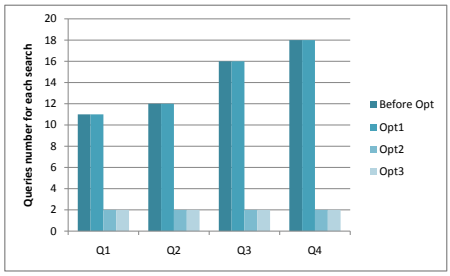

Figure 12: Queries Number.

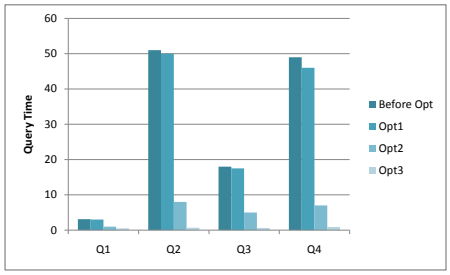

Figure 13: Query Time.

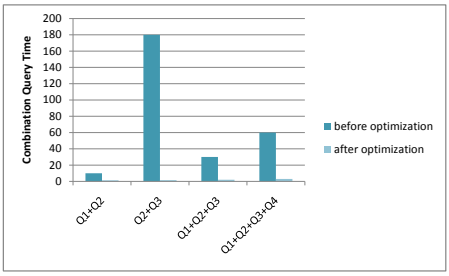

Figure 14: Combined Query Time.

Table 6: Factors on Performance

Factor1	Q1	Q2	Q3	Q4
Number	249	3109	1414	3375
Factor2	Before Opt	Opt1	Opt2	Opt3
Rows	400000	400000	12000	below 4000

Table 7: Query Time

	Q1	Q2	Q3	Q4
Before Opt	3.1s	51s	18s	49s
Opt1	3s	50s	17.5s	46s
Opt2	1s	8s	5s	7s
Opt3	0.5s	0.7s	0.6s	0.9s

Table 8: Queries number for each query

	Q1	Q2	Q3	Q4
Before Opt	11	12	16	18
Opt1	11	12	16	18
Opt2	2	2	2	2
Opt3	2	2	2	2

Table 9: Performance of Combination Query

	$Q_{1,2}$	$Q_{2,3}$	$Q_{1,2,3}$	$Q_{1,2,3,4}$
Number	108	500	56	35
Before Opt	10s	3min	30s	1min
After Opt	1.3s	1.5s	2s	3s

of existing RDF storage systems use relational DBMS, such as Jena[10], Sesame [11], 3store [12], RStar [14]. The main advantage of the approach is that the SPARQL queries are translated into equivalent SQL queries and a mature and vigorous relational query engine with transactional processing support can be reused to provide major functionalities for RDF stores. However, there are lots of join operations in real applications which make query time-consuming. As a result, both Jena and Oracle propose changes to the schema to reduce the number of joins. It has been proved that storing RDF triples by different property tables outperforms storing all triples in a single table. It also makes attribute typing possible, which saves space and can speed up certain operations such as numerical functions, aggregations, or comparison operations within index code. But this approach need the RDF data structured, otherwise it makes the tables sparse. Besides, as Wilkinson points out[5] to avoid sparsity of tables, too many property tables are created. Thus, queries requiring joins or unions to combine data from more tables which reduce the query speed. There is also a problem of the abundance of multi-valued found in RDF data with property tables . Abadi D J et al. point out SW-Store[4], which is a vertically partitioned DBMS. It can avoid sparsity of property table and solve the multi-valued problem, but increase the number of joins.

Jena is a leading Semantic Web toolkit which provides rich Model API for manipulating RDF graphs, including I/O modules for: RDF/XML, N3 and N-triple; and the query language RDQL. Users can choose to store RDF graphs in memory or in persistent stores. The Jena database subsystem implements persistence for RDF graphs using an SQL database through a JDBC connection. Jena also supports two basic schema types: both a denormalized schema used for storing generic triple statements as well as property tables to store subject-value pairs related by arbitrarily specified properties.

7. CONCLUSION AND FUTURE WORK

There is usually a promising hope that semantic technologies can be put into work to allow for smart data management and query. This paper presents *SmartHR*, a typical web semantic application for human resource management. It provides semantic interfaces for users to find the suitable candidates accurately and efficiently from massive resumes. As the basis, we build an ontology to characterize human resource management. Our main work includes designing and implementing the fundamental architecture, knowledge base construction and further performance optimization. Our work covers the common and essential techniques in semantic web, such as data model transformation, storage and search engine design. From our practice, developers can draw useful experience and conclusions.

For a long period, our future work will still focus on the performance optimization. Current query response time will be impacted negatively by the candidate number's rapid growth. Distributed systems, for example the popular NoSQL stores, or even parallel computing paradigms may be leveraged to handle the continuous growing up data. Moreover, how to extract the structured data from some unstructured files (such as images) is also worthy of study.

8. ACKNOWLEDGEMENTS

This work was partially supported by the National High-Tech Research and Development Plan of China under Grant No. 2012AA011204.

9. REFERENCES

[1] Berners-Lee, et al. The Semantic Web. Scentific American Magazine. Retrieved March 26, 2008.

[2] Klyne G, et al. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, 2004, 10.

[3] Carroll J J, et al. Jena: implementing the semantic web recommendations. WWW, 2004:75-83.

[4] Abadi D J, Marcus A, et al. SW-Store: a vertically partitioned DBMS for Semantic Web data management. VLDB Journal 2009, 18(2):385 406.

[5] Wilkinson K. Jena property table implementation. Proc of the 2nd Int Workshop on Scalable Semantic Web Knowledge Base Systens. 2006:35-46.

[6] Harris S, Shadbolt N. SPARQL query processing with conventional relational database systems. SSWS 2005.

[7] Elliott B, et al. A complete translation from SPARQL into efficient SQL. IDEAS 2009. 31 42.

[8] N. Shadbolt, T. Berners-Lee, W. Hall, The Semantic Web revisited, IEEE Intell. Syst. 21 (3) (2006) 96-101.

[9] K. Wilkinson, et al. Supporting Scalable, Persistent SemanticWeb Applications, IEEE Data Eng. Bull. 26 (4) (2003) 33-39.

[10] K. Wilkinson et al. Efficient RDF storage and retrieval in Jena2. In Proc. of the International Workshop on SWDB, 2003, pp. 131-150.

[11] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: a generic architecture for storing and querying RDF and RDF Schema. ISWC 2002, pp. 54-68.

[12] S. Harris, et al. 3store: efficient bulk RDF storage, In Proc. of the International Workshop on Practical and Scalable Semantic Systems (PSSS), 2003.

[13] Erling, O. Implementing a SPARQL compliant RDF triple store using a SQL ORDBMS. Technical Report, 2001.

[14] L. Ma, et al. RStar: an RDF Storage and Query System for Enterprise Resource Management. CIKM 2004, pp. 484-491.