Volume 2, No. 3, March 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

APPLICATION BASED SEMANTIC WEB MINING TECHNIQUE

Mahindra Pratap Singh Dohare[*1] and Sanjaydeep Singh Lodhi[*2]
Department of C.S.E. (Software System)
Samrat Ashok Technological institute
Vidisha, M.P., India.
{mps.mt32@gmail.com and sanjayeng.mt@rediffmail.com}

Vinod Mahor[*3]
Department of IT
Samrat Ashok Technological institute
Vidisha, M.P., India.
{vinodengg.mt@rediffmail.com}

Abstract-The Web is a huge read-write information space where many items such as documents, images or other multimedia can be accessed. In this context, several information technologies have been developed to help users to satisfy their searching needs on the Web, and the most used are search engines. Search engines allow users to find Web resources formulating queries (a set of terms) and reviewing a list of answers. The Semantic Web improves the Web infrastructure with formal semantics and interlinked data, enabling flexible, reusable, and open knowledge management systems. The move towards open and interlinked data on the Web and the Semantic Web results in more open systems. In contrast to traditional database-driven applications, open systems liberate the data that they operate on: sources are decentralized, data can be semi-structured with arbitrary vocabulary and contributions can be published anywhere. This thesis offers algorithms and components that simplify and support knowledge management based on Semantic Web technology. We address four areas of Semantic Web application development: programmatic access: how to program against the flexible graph-based model; data navigation: how to navigate arbitrary information spaces; data entry: how to guide users through collaborative recommendation; and data discovery: how to locate relevant data sources. Our hypothesis is that the issues of programmatic access, data navigation, data entry, and data discovery can be addressed, with acceptable results, through the sole introspection of instance data at runtime, without relying on fixed schema structures at design time. In all four areas we devise solutions that are domain independent, rely only on instance data and dynamically adjust to the available data.
Keywords- Semantic Web, Data Mining, Ontology.

INTRODUCTION

World Wide Web search services have become the most heavily used online services, with millions of searches performed each day. The only purpose of these search engines is to retrieve the exact inform action that the user wants, or a close approximation of this, from the loosely organized Internet. Inspired by the success of the Web and determined to further improve its potential, advances have been made towards a "Semantic Web": a world-wide network of simple statements, which, through its interconnections and sheer size, could serve as global knowledge management repository. The Semantic Web is now moving from a vision to a reality. The fundamental standards (RDF, RDFS, and OWL) have been developed, data is becoming available, and infrastructure is emerging. With these foundations given, we can start building applications that move towards the original vision: to improve the awareness, management and reuse of our (scientific) knowledge. The Web has indeed grown from a tool to improve scientific collaboration into an indispensable form of communication. And beyond communication, the Web is a means for information exchange and a global knowledge repository. But the reuse of information on the Web is limited since most data is hidden in databases instead of published as online interlinked resources [1] and [2].

Furthermore, most Web applications are not designed for data reuse but strictly rely on their own relational database with a fixed schema: application developers design a database schema and then, on top of that schema, construct the application logic which generates web-pages for user interaction. Such a centralized schema-centric architecture offers only limited possibilities for data integration and reuse because of schema dependency, the lack of global identifiers and the isolated nature of schemas. Changing this situation, by opening up the applications and their data, would improve knowledge management but raises several challenges: how to manage and query the web of linked data, how to align different data models and vocabularies and how to visualize and navigate the connected graph of information. The Semantic Web enables data reuse and information exchange on the Web, and, compared to traditional database-centric applications, simplifies development of such mash-ups by accommodating simple integration of data from various sources. But adopting the open and decentralized view of the Semantic Web complicates application development, since Web applications are traditionally developed using application frameworks that rely on closed systems with fixed relational schemas and centralized points of access and control.

This thesis offers algorithms and components that simplify and support knowledge management applications based on Semantic Web technology. We address Semantic Web application development in the four areas mentioned above: *programmatic access*: how to program against the flexible graph-based model; *data navigation*: how to navigate arbitrary information spaces; *data entry*: how to provide annotation support using collaborative recommendation; and

data discovery: how to locate In each area, the main obstacle of existing solutions is their dependency on fixed, a-priori, schema knowledge. Such dependency is unattainable in the world of open, interlinked, Semantic Web systems. In open systems, which acquire and integrate arbitrary data from arbitrary data providers during runtime, which operate in the decentralized environment of the Web without central management or control, without centralized guidance on data schemas and vocabularies, schema independence is crucial. In an open system, the set of schemas that will be encountered is not known at design time, and can therefore not be accounted for: if the system would be customized for a particular set of schemas, it would no longer be an open system. Furthermore, data integration on the Web involves integration of heterogeneous and widely varying schemas; after integration, the combined data no longer conforms to the original schemas, can often not be described by some fixed schema at all, and becomes "semi-structured". Therefore, open systems that integrate data on the Web, and techniques that manipulate such open data, should not rely on fixed schemas. We therefore try to address application development in these four areas through flexible, schema-independent, solutions that rely only on instance data [3] and [5].

BACKGROUND

The Semantic Web is an ongoing evolution of the Web into a more powerful and more reusable infrastructure for information sharing and knowledge management. The current Web is a publishing platform and indeed allows us to connect with arbitrary information sources across all physical and technical boundaries. But the Web is merely a publishing infrastructure of documents and links; very little consideration is given to the content or meaning of the documents or to the meaning of the links. As a consequence, the Web serves as an excellent giant document repository and, as a communication platform, enables the provision of online services, but knowledge reuse is limited because no uniform standard is available to express the meaning or intended usage of pieces of online information.

The Semantic Web is a web of information that is more understandable and more usable by machines than the current Web. It can be regarded as an extension of the existing Web, whose information is mostly human-readable. Although the current Web also has some machine-usable structure such as head and body of documents, levels of heading elements, classes of div elements, this structure has coarse granularity and little agreed-upon meaning. The Semantic Web allows for finer granularity of machine-readable information and offers mechanisms to reuse agreed-upon meaning. The Semantic Web can also be considered similar to a large online database, containing structured information that can be queried. But in contrast to traditional databases, the information can be heterogeneous: it does not conform to one single schema; the information can be contradicting: not all facts need to be consistent; the information can be incomplete: not all facts need to be known; and resources have global identifiers allowing interlinked statements to form a global "Semantic Web".

The fundamental data-model of the Semantic Web is the Resource Description Framework (RDF). RDF is a language for asserting statements about arbitrary identifiable resources. The use of global identifiers (URIs) allows statements from different sources to interlink, ultimately forming a hyper-graph of statements. RDF is a formal language in the sense that a syntax, grammar, and model-theoretic semantics are defined. The semantics provide a formal meaning to a set of statements through an interpretation function into the domain of discourse. But this interpretation function is relatively straightforward and explicit: the semantics of RDF prescribe relatively few inferences to be made from given statements; there is only little implicit information in statements. RDF can thus be seen as a language for statements without specifying the meaning of these statements [2] and [3].

More poetically, RDF can be regarded as an alphabet, allowing one to construct words and sentences, but not yet a language, since the words and sentences have not yet been given a meaning. Such computer-usable meaning can be achieved by defining a vocabulary (a set of terms) for RDF and by specifying what should be done when such a term is encountered. Currently, two such vocabularies have been agreed upon and standardized. The first is RDF Schema (RDFS), which allows one to express schema-level information such as class membership, sub-class hierarchies, class attributes (properties), and sub-property hierarchies. RDFS allows simple schema information, but its expressiveness is limited. The Web Ontology Language (OWL) therefore extends RDFS (although the two are formally not completely layered) and provides terms with additional expressiveness and meaning. Each statement in RDF is a triple of subject, predicate, and object, which can be read as "*subject* has a *predicate* with value *object*." RDF defines three types of elements: identified resources (identified by their URI), unidentified resources (blank nodes), and literals (data-values). Only resources (identified or unidentified) can be the subject of a statement; only identified resources can be the predicate of a statement; and any element can be the object of a statement. RDF is an abstract data-model and can be serialized in several formats such as RDF/XML5, N-Triples6, Turtle7, or N38 (of which only RDF/XML is officially endorsed by the W3C).

In terms of semantics, the only statements to be derived from these two triples are these two triples themselves and a set of axiomatic triples. Also note that in terms of satisfiability (a graph is satisfiable if it has a model) every RDF graph is satisfiable: every graph in RDF is true since it is not possible to express any contradictions. A set of RDF statements can be regarded as a (labelled) graph: subjects and objects are nodes and predicates are edges. Since the predicates can appear as subjects themselves, RDF statements can more precisely be regarded as a hyper-graph. But a set of RDF statements is not exactly a graph, since RDF statements have additional semantics that a graph as such does not have [1] and [4] and [15].

Data models and query languages-

The network and hierarchical models, initial representations of large-scale data, had a low abstraction level and little flexibility. The relational model, described by Codd and implemented by all relational databases, introduced a higher level of abstraction by separating the logical from the physical data levels. In contrast to RDF, the relational model

assumes a fixed and a-priori defined data schema; furthermore all data and schema elements use local identifiers, which hampers data reuse, integration and extensibility. Semantic models such as the entity-relationship model, increase the level of abstraction and allow data modelers to include richer schema semantics such as aggregation, instantiation and inheritance.

The object-oriented data model aim to overcome limitations of the relational model (type definitions limited to simple data types, all tuples must conform to schema structure, which cannot be modified during runtime, limited expressive power in e.g. inheritance or aggregation) through the principles of object-oriented design. Semi-structured data is self-describing in the sense that the schema information (if available) is contained in the data itself; the data schema defines relatively loose constraints on the data or is not defined at all. Semi-structured data can be generally characterized as follows: it may have irregular structure, in which data elements may be heterogeneous, incomplete, or richly annotated; it may have implicit structure, in which data elements may contain structures or even plain-text that need to be post-processed; the structure may be used as a posterior data guide instead of an a-priori constraining schema; the structure may be rapidly evolving; and the distinction between data and schema may not always be clear, in the presence of evolving schemas, weak constraints, and the meta-modeling capabilities of graphs and hyper-graph models of semi-structured data.

RDF is based on semi-structured data models but differs in the expressiveness of its schema language, in the existence of blank nodes, and in the fact that edge labels (predicates) can be resources themselves and thus form a hyper-graph. XML, with its XSD schema language, can also be considered as a semi-structured model. Important differences between RDF and XML are, on the data level, the universality of the hyper-graph structure of RDF versus the tree structure of XML. On the schema level, the higher expressiveness of RDFS versus XSD with respect to class membership, class and property inheritance and conjunctive classes [5] and [6] and [16].

Characterizing Semantic Web applications-

The relation between the Web and the Semantic Web has changed, as the understanding and interpretation of the Semantic Web has evolved over time: on the one hand, the vision of the Semantic Web has been interpreted as an enrichment of the current Web, employing for example named-entity recognition or document classification, resulting in semantically annotated Web documents on the other hand, the Semantic Web has been interpreted as an interlinked "Web of data"; enabling ubiquitous data access and unexpected reuse and integration of online data sources.

We focus mostly on the latter interpretation and consider a Semantic Web application to be an application that delivers some functionality to its users while using Web standards such as HTML, CSS, and JavaScript for its user interface, using Web standards such as HTTP to deliver the application to its users and using information from online data sources, using Semantic Web standards such as RDF(S), OWL, and SPARQL.

This Web of data has been referred to as "Web 3.0": the continuing evolution of the Web towards the usage of open, interchangeable, data – even though little consensus exists on the evolution from "Web 1.0" to "Web 2.0" which has been characterized as the advent of social Web applications, of rich user interfaces, of mashups and data exchange, of harnessing the controversial wisdom of the crowds, of business models that build extensible platforms rather than closed applications and as combinations of all these characteristics. Following the notion of "Web 3.0," much of the application infrastructure and development approach can be shared between existing Web applications and Semantic Web applications, since both are open architectures for information sharing: the first oriented more towards documents, the second more towards data. Both are decentralized, heterogeneous, with freedom of publishing, allowing anyone to create documents or assert statements at any location, using any vocabulary or structure [7] and [8] and [10].

The Web is not only a publishing infrastructure but also an application platform for Web applications. But existing applications use the Web primarily as a means to access their application, generating HTML pages from their database content and serving these pages over HTTP. These database-driven applications result in a "deep" or "hidden Web", whose dynamically-generated pages do not conform to traditional Web principles such as hyperlinks and are thus hard to crawl and index.

Web Application	Semantic Web Application
centralized	Decentralized
one fixed schema	semi-structured
one fixed vocabulary	arbitrary vocabulary
centralized publishing	publish anywhere
one data source	many distributed data sources
closed systems	open systems

Table1: Traditional vs. Semantic Web applications

More importantly, these database-driven applications are closed systems that rely on a single centralized data source; due to the inherent limitations of their relational databases, these Web applications operate on fixed data structures and schema, use one fixed vocabulary, and do not interlink their data. In contrast, Semantic Web applications are more aligned with the principles of the Web, such as interoperability, universality, evolvability and decentralization, hence the incremental version number of "Web 3.0". As shown in Table, Semantic Web applications are decentralized and open, operate on distributed data that can be published anywhere, may conform to arbitrary vocabulary and follow semi-structured schemas.

Scenario: exploring interlinked online communities to illustrate the relation between existing Web application and Semantic Web applications and the notion of open systems and decentralized data, we consider the development of an application for browsing and collecting information from online social communities.

Online communities are sites such as forums, weblogs, mailing lists or IRC channels. Some of these sites are more centralized, such as forums or bulletin boards, while others such as weblogs or IRC channels are more decentralized and disparate. But from an abstract perspective all such communities are relatively similar: they allow users to group themselves online and exchange and discuss about their particular topics of interest. Often, discussions range over

several of these communication channels. For example, to solve an installation problem of a wireless card in the Ubuntu Linux distribution, a user should search the Ubuntu community forums for some helpful advice but also look on weblogs and the Ubuntu-users mailing list.

Currently, users have to browse these communication channels manually and repeat their query in various different systems: the forum software, the mailing list online archives, a weblog search engine, etc. Search engines help to find individual posts but do not allow browsing across various online communities. For the end-user, it would be convenient if all these community sites were collected in a single place [11] and [12].

The Semantic Web enables such community sites and other information publishers to act as open systems; it also enables application developers to build open systems that reuse the information published by the providers. Opening up traditional database-driven Web applications without Semantic Web technologies would be difficult, due to the isolated nature of relational databases:

Strict schemas: since relational databases rely on strict schema information and restrict their instances to the schema, data integration without a predefined common schema is complicated.

Local identifiers: since database is traditionally used as self-contained sources, they use local identifiers instead of global identifiers. The absence of global identifiers complicates integration since shared concepts and instances are harder to identify, well-known in the database community as the "record linkage" problem.

Isolated vocabulary: since database vocabulary such as table names and column names are self-contained and cannot extend or reuse other vocabulary, schema elements cannot be related to elements from other database schemas, thus requiring intricate schema matching and schema alignment techniques during data integration [1] and [13] and [14].

PROPOSED TECHNIQUE

Programmatic Access to Semantic Web Data-

Any application needs to access data sources to retrieve, manipulate, and display data to its users. In traditional relational-database applications, various solutions have been developed that offer programmatic access to relational data sources.

But these existing mapping approaches do not suffice for Semantic Web applications because: (i) the access and manipulation patterns differ from the relational setting, and (ii) the conceptual model of Semantic Web data and the semantics of RDF Schema differ substantially from both the object-oriented paradigm and the relational paradigm, on which the existing mappings rely. To support application developers new mappings need to be developed that provide programmatic access to Semantic Web data and offer the access patterns required by typical applications. Semantic Web applications share large portions of functionality with traditional Web applications, such as authentication management, session management, caching, user interface widgets, reusing these Web application frameworks is desirable. But since these frameworks rely on an object–relational mapping, a similar mapping from graph-based Semantic Web data to programmatic objects would be required. Having analysed the typical access patterns for a mapping library and explained the suitability of implementing such mappings in a dynamically-typed programming language, we now present Active RDF, an object-oriented API for Semantic Web data. Active RDF maps RDF Schema classes to programming classes, RDF resources to programming objects and RDF predicates to methods on those objects, thus lifting data elements into first-class citizens (objects of the language itself), the general principle of ActiveRDF is to represent RDF resources through transparent proxy objects. Each proxy object represents one RDF resource but does not contain any state. All methods on the proxy object are translated into read or write queries related to the proxy's RDF resource. Our architecture consists of four layers which incrementally abstract RDF data into objects, as shown in Figure. Such a layered architecture supports (i) design based on increasing levels of abstraction, allowing the implementation to partition the problem into a set of incremental steps; (ii) gradual enhancement because as each layer interacts only with the layers directly above and below itself, the effects of changes are limited; and (iii) reuse, since different implementations of the same layer can be used interchangeably. The layers incrementally abstract the RDF statements from the data sources into objects: Adapters provide access to a specific RDF data-store by translating generic RDF operations to a store-specific API. Such RDF data-store specific adapters are necessary, because of the absence of a general standardized query language which provides create, read, update,

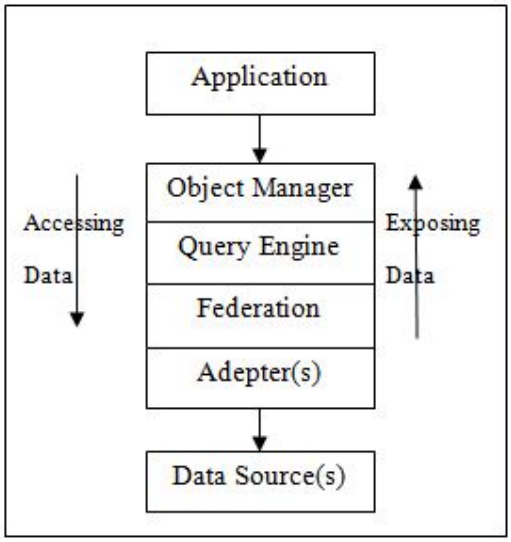

Figure1: ActiveRDF architecture

and delete access. The adapter layer assures vendor-independence in the rest of the architecture, as all store-specific operations are encapsulated within the adapters. The adapters translate and execute queries from the federation manager into a query language supported by their data source.

Extending faceted navigation for semantic web data-

The second element of Web application development is data navigation in the user interface. Since open Semantic Web

systems integrate data from decentralized sources that are not under their control, they cannot assume a single fixed schema to apply for all encountered data. Application developers should therefore not rely on predefined user interfaces to browse that data, since a predefined user interface would not be able to adjust to newly encountered information.
Our formal model for faceted navigation of Semantic Web data; we first add several operators to the ones described above and show how we have implemented these in our prototype interface. We then introduce an initial approach to automatically ranking and selecting useful facets at runtime, using metrics based on the available instance data.
Users can browse the dataset by constraining one or several of these facets. At the top-centre of the screenshot we see that the user constrained the dataset to all fugitives and in the middle of the interface we see that three people have been found conforming to that constraint.

Join selection Given that RDF data forms a graph, we often want to select some resources based on the properties of the nodes that they are connected to. For example, we are looking for "all posts created by somebody, who in turn knows somebody who works in DERI", as shown in Figure c. using the join-operator recursively, we can create a path of arbitrary length, where joins can occur on arbitrary predicates. In the interface, the user first selects a facet (on the left- hand side), and then in turn restricts the facet of that resource. In the given example, the user would first click on "creator", then click on "knows" and then click on "workplace", and only then select the value "DERI". Intersection As in the Flamenco interface, multiple selection operators are automatically evaluated in conjunction, by applying the intersection operator. For example, we can combine the three previous examples to restrict the resources to "all untitled posts about cars written by somebody who knows somebody in DERI", as shown in Figure above. This operator is not explicitly available in the interface since it is applied automatically on multiple selections.
Inverse selection All operators have an inverse version that selects resources by their inverse properties.

Algorithms for recommendation of Semantic Web vocabulary-
The third challenge of Semantic Web application development that we address arises during data entry: the need to guide users when creating Semantic Web data into a meaningful information space. For contributions to be understood by others, existing vocabulary should be reused where possible, which can be achieved by offering vocabulary recommendations to users while they are creating Semantic Web data.
In general, the Semantic Web supports user-generated content, since the data model and semantics allow arbitrary statements without the need to conform to predefined schemas. But provided content is still only useful as far as others understand it. A centralized policy prevents terminology divergence but would restrict users needlessly. We follow a similar approach towards a collaborative recommendation system for Semantic Web vocabulary: our "potentially overwhelming set of choices" is formed by the vocabularies (ontologies or schemas) that are available to users and that they need to decide upon while they are creating Semantic Web data. More specifically, we do not construct a complete recommender system but investigate recommendation algorithms for suggesting relevant and frequent terminology when creating annotations. We present two domain-independent algorithms that recommend vocabulary based on statistical dataset analysis.

The first algorithm is intuitive and precise, based on an explicit measure of similarity between resources. However, the similarity algorithm is not efficient (quadratic in the number of resources) since it computes similarity between all resources. The second algorithm uses an approximation of resource similarity (namely pair wise predicate co-occurrence) to achieve much improved runtime performance. Our hypothesis is, that approximating resource similarity through pair wise predicate co-occurrence yields good results, which is indeed supported by the high quality of the second algorithm compared to the first algorithm.
Algorithm 1: suggestions using resource similarity-
The task of the suggestion algorithm is to find, for a certain resource in focus, predicates to further describe that resource. The general idea of the classification-based algorithm is to divide the knowledge base in two groups, those similar to the current resource and those not similar, and to suggest the frequently occurring predicates from the similar group. For example, Figure below shows a simple knowledge base with three resources: the person "John", with his name, some friends, and homepage, the book "The Pelican Brief", with its title and author, and the person "Stefan", with his name. We want to suggest relevant predicates for "Stefan" based only on the given graph.
The algorithm consists of two steps, as shown figure below. In the first step, we divide all existing resources in the knowledge base into two sets, the similar and dissimilar ones. In the second step, we look at all predicates from the similar group and rank them using a ranking function. In the remainder of this subsection, we explore each step in more detail: how to define similarity between resources, and how to rank the selected predicates.
However, in practice we cannot ignore lookup performance on large datasets. To compute similarity, we need to look up all predicates of each resource. Depending on the lookup performance of the used data store, this could cause the whole algorithm to run logarithmic or even quadratic to the size of the dataset, rendering the algorithm impractical for reasonably large datasets. A simple solution would be to materialise the similarity between resources in memory, obliterating the need for data lookup during suggestion time. Direct materialisation however has two problems: the required memory space would be quadratic in the size of the dataset, and updating one resource would require recalculation of all similarity values with respect to this resource.
The next algorithm remedies exactly this problem and allows materialization without large memory requirements.

Algorithm 2: approximate similarity-using co-occurrence-
The general idea of the co-occurrence-based algorithm is to reduce the data from which suggestions are made, by approximating resource similarity through the co-occurrence of predicates.

We then further reduce the required space by not considering the complete power set over all predicates, but instead approximate full co-occurrence through binary co-occurrences. Most datasets contain far less unique predicates than unique resources: trivially, since the set of predicates is a subset of the set of resources, but also significantly, since most datasets use only a small number of unique predicates. As a result, materializing pair wise predicate co-occurrence requires less space than pair wise resource similarity: $O(p^2) < O(r^2)$. We thus consider only pair wise occurrences of predicates, suggest predicate candidates for each pair wise occurrence, and combine these candidates through intersection.

We therefore make two assumptions on the probabilistic model of the dataset: (1) that predicate co-occurrence correlates with resource similarity, and (2) that considering binary predicate co-occurrences to be independent events (which they are not) yields acceptable predictions. The latter allows us to pair wise consider binary co-occurrences instead of all permutations. Our algorithm is based on association rule mining used for recommendations in e.g. online stores: when buying one book, other books that are often bought together with this book are recommended. In our case, books are replaced by predicates and shopping transactions by resources.

Discovery architecture for interlinked Semantic Web data-

The final element of Semantic Web application development that we address is the runtime discovery of relevant data sources. To achieve a high information quality, applications should typically integrate data from many decentralized sources, but discovering those sources is not trivial. Due to the required network bandwidth and data storage, a Semantic Web discovery service should not be implemented by each application individually, but should instead run as an independent service and be integrated into client applications.

Architecture design

The architecture consists of several independent components that operate in several pipelines to

achieve crawling, indexing, and querying. The Web front-end is the main entry point, divided in a user interface for human access and an HTTP API for machine access. Several components are responsible for crawling and indexing RDF documents. A crawler autonomously harvests RDF data from the Web and adds found documents to the indexing queue; documents are also added to the queue when pinged explicitly through the front-end. The gatekeeper evaluates each entry in the queue and decides whether, and with which priority, we want to index it, based on whether we have seen the document before, its last modification date, its content digest, etc. The indexer extracts URIs, IFPs and keywords from each document using the reasoner and adds these to their respective index. During lookup, the interface components only need to pass the queries to the relevant index, gather the results, and generate the required output such as HTML pages with appropriate layout. As mentioned before, the whole execution pipeline and all its components, except the front-end and the inverted index, are distributed over arbitrary parallel nodes, a parallel architecture. The three indices store occurrences of resource URIs, resource IFPs and literals in RDF documents. The URI index contains an entry for each resource URI that lists the document URLs where this resource occurs. The IFP index is similar, except that instead of explicit resource URIs, the uniquely identifying pair (property; value) is used as index key, again pointing to a list of document URLs where this pair occurs. This index allows lookup of resources with different URIs that actually identifies the same real-world thing. The literal index contains an entry for each token (extracted from the literals in the documents), again pointing to a list of document URLs. In designing the index, we optimize for disk space and lookup times. Since the only required access pattern is from resource to mentioning sources, an inverted index of URI occurrences in documents is a natural structure. In general, lookup on such an index can be performed in almost constant time over the size of the index. Lookups that return a large list of documents cause longer query times, especially ontology classes. The straightforward solution is to either eliminate these occurrences as stop-words or to return only a limited set of results. Technically these indices have been implemented both as an on-disk persistent hashtable, mapping from resource URIs to mentioning documents, and in the Solr54 information retrieval engine. The on-disk hashtable is conceptually simple but less efficient in practice because it lacks standard information retrieval optimisations such as distributed indexing, efficient sort algorithms, index compression and caching. Before detailing the internals of Sindice, we analyze its feasibility. We analyze a representative sample of Semantic Web data and analyze graph-theoretical properties that allow us to predict the required index size using inverted index structures. We can then cluster and replicate such index structures to provide service in a round-robin fashion.

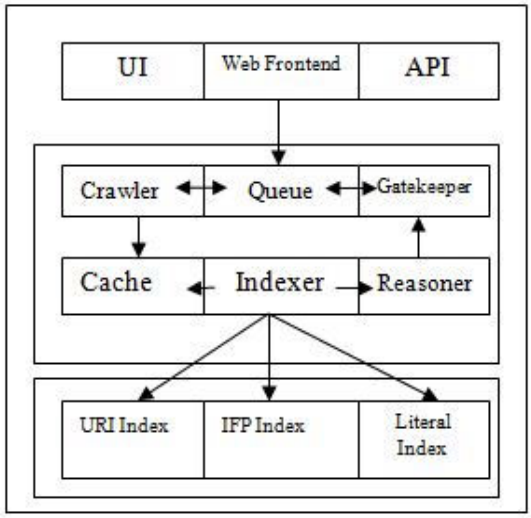

Figure2: Sindice architecture

To achieve scalability, the architecture is based on Hadoop, an existing parallel processing infrastructure, allowing us to efficiently distribute index construction over an arbitrary number of nodes. The inverted index itself has been implemented twice, once directly using persistent hash tables, an off-the-shelf information retrieval index. Since the storage requirements of both implementations rely on the reuse of resource identifiers; we have experimentally showed that this ratio of URIs/URLs follows a power law

and thus exhibits scale invariance, allowing us to confidently estimate the required data storage as a function of the number of indexed documents.

RESULTS

As any application needs to manipulate data, typical access patterns should be abstracted into higher-level libraries and embedded into the application-programming environment. We have analysed which access patterns should be supported in a programmatic manner, formally expressed these as a subset of PathLog and explained why the techniques used in traditional objectrelational mapping approaches are not sufficient for Semantic Web data. We have showed why dynamically-typed object-oriented languages offer a suitable environment for our mapping solution, given their dynamic typing of objects, which maps well onto the RDF(S) class membership, their support for full reflection, which allows us to implement the multi-inheritance of RDF(S), and their relaxation of strict object conformance to class definitions.

The figure of Querying Sesame in ActiveRDF shows the average response time (including result parsing in Java and ActiveRDF) of each query using curl, Java, and ActiveRDF in a logarithmic scale. It can be seen that for most queries ActiveRDF adds only little overhead. On some queries ActiveRDF seems to perform faster than using curl HTTP, which is probably due to random hardware variations and measurement difficulties in those small response time ranges.

Faceted browsing is a data exploration technique for large datasets. Our technique can be employed for arbitrary semi-structured content. We have presented a novel analysis of existing interfaces and have extended their expressiveness; we have also developed initial metrics for automatic facet ranking, resulting in an automatically constructed faceted interface for arbitrary semi-structured data. Our faceted navigation has improved query expressiveness over existing approaches and experimental evaluation shows better usability than current interfaces.

Our evaluation combines both the information-retrieval and the machine-learning approach: we show both precision and recall ratings and evaluate our approach using training/testing datasets through a commonly applied technique of evaluating prediction of deleted values from existing data.Our primary evaluation technique is prediction of deleted values: we pick a random resource from the dataset as a candidate for which further predicates should be suggested. We then randomly remove one or more statements about this candidate and analyse if and at which rank position the removed predicates are re-suggested. Repeated over n random resources this yields the *average resuggestion rate* (how often was the deleted predicate resuggested), the *empty suggestion rate* (how often were no suggestions given), and the *average rank* of the resuggested predicate. Since in practice not all suggestions can be displayed or will be considered by the user, We also show how many of the predicates were resuggested within the top-k of suggestions. Secondly, we measure suggestion precision (how many suggestions are valid) and recall (how many valid suggestions have we missed) based on the schema definition: we define “valid” predicates as those predicates that, according to the schema, fall within the domain of the selected candidate.

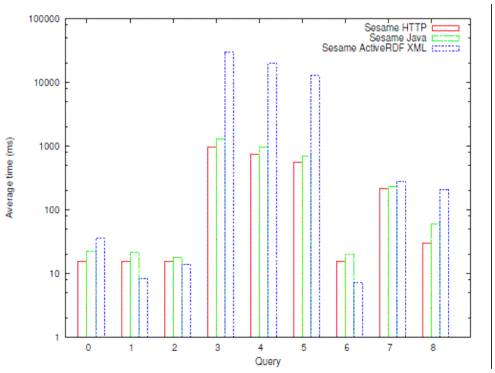

Figure4: Querying Sesame in ActiveRDF

For recall computation, we consider only predicates that are actually used in the dataset; since the algorithm considers only instance data, unused predicates are unattainable.

To achieve scalability, the Sindice architecture is based on Hadoop, an existing parallel processing infrastructure, allowing us to efficiently distribute index construction over an arbitrary number of nodes.

The inverted index itself has been implemented twice, once directly using on disk persistent hash tables, an off-the-shelf information retrieval index. Since the storage requirements of both implementations rely on the reuse of resource identifiers; we have experimentally showed that this ratio of URIs/URLs follows a power law and thus exhibits scale invariance, allowing us to confidently estimate the required data storage as a function of the number of indexed documents.

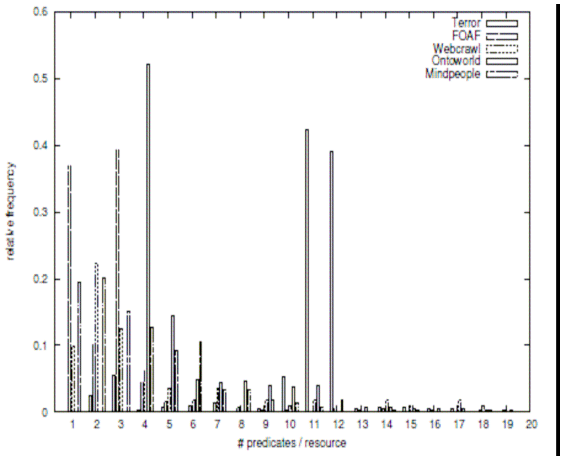

Figure3: Distribution of predicates over resources in different datasets

CONCLUSION

We have focused on four requirements in Semantic Web application development and developed algorithms and components that address these. The first requirement is to provide programmatic access to Semantic Web data embedded in the application programming language, while considering the semantic mismatches between the prevalent object-oriented paradigm and the graph-based, semi-structured, RDF(S) data model. The second requirement is the development of user interfaces, particularly for navigation of a data set, considering that Semantic Web data

can have arbitrary structure and content. We have addressed these problems subsequently in the core part of the thesis, and presented algorithms, components and implementations that support application developers. Contributions we have presented ActiveRDF, an object-oriented library for RDF data written in Ruby. We have analysed which common data patterns should be supported, why the techniques used in traditional object-relational mapping approaches are insufficient for Semantic Web data, and why dynamically-typed programming languages are well-suited to provide such language embedded programmatic data access. ActiveRDF provides a domain-specific manipulation language based on the actual available instance data, is embedded into the Ruby programming language, and is vendor-independent with respect to data stores. Additionally, ActiveRDF can serve as data layer in Ruby on Rails, providing a solution for rapid development of Semantic Web applications.

We have presented Suggest RDF, a suggestion system for Semantic Web vocabulary. We introduced two algorithms for suggesting possible predicates based on statistical data analysis. The first algorithm is based on resource similarity, achieving relatively good quality but with high computational costs. The second algorithm approximates resource similarity through pair wise predicate co-occurrence. Treating predicate occurrences as independent events simplifies computation and allows for memory-efficient materialisation, while still resulting in high quality suggestions. The materialisation scales linearly with the size of the dataset and allows for incremental updates; after materialisation, suggestion time is constant.

Finally, we have presented Sindice, an indexing and lookup service for Semantic Web data sources. Sindice allows application developers to easily discover relevant data sources for their application. Lookups for data sources can be performed directly using resource URIs, indirectly through uniquely-identifying inverse functional properties, and through a full-text search over the literals. We have analyzed several design considerations for developing such a lookup service and explained our choices in the Sindice implementation.

REFERENCES

[1] WANG Yong-gui and JIA Zhen, “Research on Semantic Web Mining”, IEEE 2010, International Conference On Computer Design And Appliations (ICCDA 2010).

[2] MohammadReza Keyvanpour, Hamed Hassanzadeh and Babak Mo hammadizadeh, “Comparative Classification of Semantic Web Challenges and Data Mining Techniques”, IEEE 2009 International Conference on Web Information Systems and Mining.

[3] Li Yu and Qiang Li, “A Novel Web Text Mining Method based on Semantic Polarity Analysis”, IEEE2009.

[4] Huimin Wang, Guihua Nie and Kui Fu, “Distributed data mining based on semantic web and grid”, IEEE 2009 International Conference on Computational Intelligence and Natural Computing.

[5] Shahab Bayati, Ali Farahmand Nejad, Sadegh Kharazmi and Ardeshir Bahreininejad, “Using Association Rule Mining to Improve Semantic Web Services Composition Performance”, IEEE 2009.

[6] Shizhan Chen, Zhiyong Feng, Hui Wang and Tao Wang, “Building the Semantic Relations-Based Web Services Registry through Services Mining”, 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science.

[7] YANG Xiao-qin, JU Shiguang and CAO Qinghuang, “A Deep Web Complex Matching Method based on Association Mining and Semantic Clustering”, IEEE 2009 Sixth Web Information Systems and Applications Conference.

[8] Nizar R. Mabroukeh and C. I. Ezeife, “Semantic-rich Markov Models for Web Prefetching”, 2009 IEEE International Conference on Data Mining Workshops.

[9] Suleyman Salin and Pinar Senkul, “Using Semantic Information for Web Usage Mining Based Recommendation”, IEEE2009.

[10] Yi Feng, “Towards Knowledge Discovery in Semantic Era”, 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2010).

[11] Elaheh Momeni, “Towards (Semi-)Automatic Moderation of Social Web Annotations”, IEEE 2010 International Conference on Social Computing / IEEE International Conference on Privacy, Security, Risk and Trust.

[12] Julia Hoxha and Sudhir Agarwal, “Semi-automatic Acquisition of Semantic Descriptions of Processes in theWeb”, IEEE 2010 International Conference on Web Intelligence and Intelligent Agent Technology.

[13] Guiguang Ding and NaXu, “Automatic semantic annotation of images based on Web data”, 2010 Sixth International Conference on Information Assurance and Security.

[14] Zhu Jiang, Jiang jun and Li san-ping, “A Semantic Adaptation Method and System”, IEEE 2010, International Forum on Information Technology and Applications.

[15] HongLiu and XiaoHongYu, “Application research of Semantic Ontology technology in Content-Based Image Retrieval”, 201O International Conference On Computer Design And Appliations.

[16] Lorand Dali and Dunja Mladenic, “Visualization of Web Page Content Using Semantic Technologies”, IEEE 2010 14th International Conference Information Visualisation.

