ECAI
2012

Twenty European Conference
on **Artificial Intelligence**

Montpellier, August 27th 2012

1st International Workshop on Artificial Intelligence meets the Web of Data

AImWD 2012

Christophe Gueret
Vrije Universiteit Amsterdam

Dino Ienco
IRSTEA Montpellier

Francois Sharffle
LIRMM Montpellier

Serena Villata
INRIA Sophia Antipolis

The First Workshop on Artificial Intelligence meets the Web of Data

The Linked Data initiative aims at improving data publication on the Web, thereby creating a "Web of Data": an interconnected, distributed, *global data space*. The Web of Data enables people to share structured data on the Web as easily as they can currently share documents on the Web of Documents (WWW). The basic assumption behind the Web of Data is that the value and usefulness of data increases with the amount of interlinking with other data. The emerging Web of Data includes datasets as extensive and diverse as DBpedia, Flickr, and DBLP. A "tip of the iceberg" representation of its content is behind maintained at http://lod-cloud.net

The availability of this global data space creates new opportunities for the exploitation of Artificial Intelligence techniques in relation with knowledge representation, information extraction, information integration, and intelligent agents. Two approaches can emerge: (i) using Artificial Intelligence techniques to address the problems the Web of Data faces or, (ii) using the design principles of the Web of Data to improve knowledge representation within Artificial Intelligence techniques.

The workshop brings together researchers and practitioners working on the Web of Data and/or Artificial Intelligence to discuss the union of these two research areas. Several core challenges, such as the interconnection of heterogeneous datasets, the provenance of the information and trust issues are at the centre of the discussion. With this workshop, our goal is to contribute to the birth of a community having a shared interest around publishing data on the Web and exploring it using AI technologies – or the inverse, developing and improving Artificial Intelligence technologies which use tools from the Web of Data.

Christophe Guéret, Dino Ienco, François Scharffe and Serena Villata

Workshop Chairs:
Christophe Guéret, Vrije Universiteit Amsterdam
Dino Ienco, IRSTEA Montpellier
François Scharffe, LIRMM Montpellier
Serena Villata, INRIA Sophia Antipolis

Program Committee

Martin Atzmueller	University of Kassel
Dominik Benz	University of Kassel
Bettina Berendt	K.U. Leuven
Antonis Bikakis	University College London
Paolo Bouquet	University of Trento (Italy)
Madalina Croitoru	LIRMM, Univ. Montpellier II
Richard Cyganiak	Digital Enterprise Research Institute, NUI Galway
Jerome David	INRIA Rhône-Alpes
Gianluca Demartini	University of Fribourg
Peter Edwards	University of Aberdeen
Nicola Fanizzi	Dipartimento di Informatica, Università di Bari
Fabien Gandon	INRIA
Guido Governatori	NICTA
Christophe Gueret	Vrije Universiteit Amsterdam
Harry Halpin	University of Edinburgh
Dino Ienco	University of Torino
Michel Leclère	LIRMM (CNRS - UM2)
Roberto Navigli	Sapienza Universita' di Roma
Matteo Palmonari	University of Milano-Bicocca
Domenico Redavid	University of Bari
Francois Scharffe	LIRMM, University of Montpellier
Andrea G. B. Tettamanzi	Università degli Studi di Milano
Serena Villata	INRIA Sophia Antipolis
Jun Zhao	University of Oxford

Table of Contents

Author Index

Vocabutek: An Electronic Auction of Knowledge for Linked Data

Harry Halpin[1] and **Ron Reck**[2]

Abstract.

In this paper, we present ongoing work on Vocabutek, an online electronic auction for knowledge in the form of Semantic Web vocabularies as used in Linked Data. Our online service plans to solve one of the most long-standing problems on the Semantic Web: How can users of the Semantic Web easily find and create vocabularies that satisfy their needs? Outside of work in Semantic Web search engines, there has been little work in making it easy for users to specify what kinds of requirements they have for expressing vocabularies and communicating those needs directly to people who create vocabularies. In Vocabutek, a consumer specifies their information need for a vocabulary in the form of constraints over multi-valued attributes. Then the service, which maintains an index of vocabularies, uses the these constraints to match vocabularies produced by providers that may fully or even partially match those constraints. In the case of where there is not a match it can prompt providers to create a new vocabulary to match the constraints. Lastly, an auction can fully or partially automate the determination of the cost of vocabularies that match the constraints expressed by the user.

Despite the large amount of work on the foundations of the Semantic Web over the last decade, there remains a gaping hole in the entire stack: There is no way for consumers of Semantic Web technology to easily discover whether or not a Semantic Web vocabulary that is suitable for their particular need exists. Even more importantly, assuming that there does *not* exist a vocabulary that matches the needs of a user, there exists no way to communicate this to the community of people who create Semantic Web vocabularies. Furthermore, even if these needs exist, there is little incentive to create vocabularies, and even less to maintain them.

A quick aside on terminology: We use the term *vocabulary* in a more broad-ranging way that is often conceived, to include almost all digital information produced primarily to fulfill some information need. This includes both RDF Schema, SKOS, and OWL, and all non-Semantic Web vocabularies on Vocabutek are given a default Semantic Web form so they can be indexed and stored by our Sesame triple-store. A *category* is metadata about the subject of the vocabulary. *Attributes* describe any aspect of the vocabulary, including metadata about its provenance and maintenance. A vocabulary consists of *terms* with a structure, where the terms may be terms in a natural language or a machine-processable language, or data such as numeric and image data. Therefore, we do not restrict the notion of vocabulary to be RDF Schemas or OWL *TBox*, but allow instance data (*ABox*) in a vocabulary. An *information need* is the desire for information that can be satisfied by a vocabulary. For example, a list of all the names of gangs with a taxonomy of their kinds of relationships could fulfill the information need of a law enforcement agency, and a list of all the parts of a kind of antique bicycle could fulfill the need of an bicycling enthusiast attempting to rebuild an old bicycle. *Consumers* are users of vocabularies, usually individual humans or humans working on the bequest of an organization, while *providers* are those who can either produce or provide vocabularies that satisfy the information need of one or more consumers, usually because they are domain experts in an area.

[1] W3C/Institut Recherce et la innovation du Centre Pompidou, email: hhalpin@w3.org

[2] RReckTek LLC, email:rreck@rrecktek.com

1 The Vocabulary Crisis

The first issue one encounters using the Semantic Web - and in information integration in general - is an inability to discover accurate and well-maintained vocabularies for their particular problem or domain. This 'vocabulary crisis' undermines the confidence of the wider information technology world in the Semantic Web, since the entire Semantic Web crucially relies upon distributed vocabularies given URIs to share information. There are a number of long-standing centralized vocabulary hosting sites that offer indexes, such as *rdfs.org* and *vocab.org*, to locally hosted sites, or sites such as *semanticweb.org* that offer pointers to popular vocabularies hosted by other sties. However, to a large extent, none of these vocabulary directories have very large coverage or are even well-maintained. Therefore, currently the common practice is to use a Semantic Web search engine like FALCON-S Concept Search [1] and to type in keywords in order to locate a Semantic Web vocabulary in the domain of interest. This particular technique often is very limited, as it produces too many results for a human to feasibly go through, particularly as displaying and parsing Semantic Web vocabularies is difficult for the ordinary, even technically competent, user. As shown in previous experiments having deploying humans to rank for relevance rankings to the Semantic Web, even with commonly used key-words describing general purpose entities and concepts, currently it appears that Semantic Web search engines can only find relevant information approximately for two-thirds of queries, and of those relevant vocabularies, approximately half the time the relevant vocabulary is not the first vocabulary returned by the Semantic Web search engine [2]. The original Semantic Web vision expected that, in general, ordinary end-users and organizations would create their own vocabularies if they could not find them otherwise, such as by using a Semantic Web search engine. This is possible because there exists a number of ontology creation tools exist like Protege [4]. However, many users find these tools hard to use, as these are tools built for the 'working ontologist' and not the ordinary end-user who has some information need he would simply like to easily specify. Also, those with specific domain expertise may not be versed in ontology creation. Lastly, none of these tools offer an easy-to-use and well-maintained Web-based

interface for the creation of vocabularies, instead relying on users to install large programs locally, which historically users are unwilling to do. While tools that offer a Web-based form interface like Neologism from DERI[3] and the commercial service *knoodl.org* promise to help satisfy this need, Neologism is still under development and not for public use, while *knoodl.org* does not let users download and store their vocabulary elsewhere. Combined with the recent technical problems of long-standing pieces of Semantic Web infrastructure like *purl.org* and the FOAF documentation, it is clear that more effort must be made to provide some some guarantee of stability and maintenance of the vocabulary, particularly for users like businesses and governments, as well as clear legal guidelines for vocabulary use and patents. This general vocabulary hosting disaster around the Semantic Web has led commercial companies like Google who are investing in RDFa to focus on creating and hosting their own URIS[4] rather than use any of the Semantic Web community-driven approaches.

2 Monetary Incentives for Vocabularies

Most importantly, the 'vocabulary crisis' exists because there is no coherent incentives for the creation of vocabularies. Yet there are few things more valuable in the information economy than knowledge. Ranging from general purpose vocabularies describing geographical locations to up-to-date descriptions of the MP3 players, accurate and well-maintained knowledge may very well have a *monetary* worth. Therefore, we hypothesize that consumers of information may very well *pay* both domain experts and experts in vocabulary some monetary value for accurate vocabularies. There are few clearer incentives for the creation and maintenance of vocabularies. To us this seems more realistic than attempts to like OntoGame, inspired by the work of von Ahn, to make ontology creation a 'game' [6]. We also consider this market-driven approach a likely superior approach for determining the monetary worth of vocabulary rather than methods like ONTOCOM, that provide a an approach to determining ontology cost that factors in projections such as personnel and project overhead. While ONTOCOM is a valuable approach, the financial value of vocabularies is an inherently subjective phenomenon that can best be determined by market forces rather than either a-priori attempting to estimate the value of ontologies. The real problem is finding a way to bring together vocabulary providers and consumers and a way to let them jointly determine the monetary price of an existing vocabulary or a new vocabulary.

The most visible online focus for the commercial vocabulary community is arguably *http://www.taxonomywarehouse.com*, a website operated by the Dow Jones company Factiva. As of July 2009 they have listings for almost seven hundred taxonomies and thesauri, which are categorized into at least one hundred different subject domains. Each vocabulary listed on the website may be described by approximately twenty metadata elements including the type of vocabulary, a description, number of terms, types of relationships, languages, available formats, and URIs that point to more information. However, the vocabularies themselves are given only a set price, and there is little incentive or way for users of Taxonomy Warehouse to determine what a realistic market-value for their vocabulary is. Thus, there are only a handful of companies that actively market knowledge collections, the most notable is Intellisophic who creates vocabularies via an patented automatic computer program, but there is often problems of validation and quality. Other companies like Gail specialize in hand-made vocabularies for particular institutional needs, like educational vocabularies. The type of challenges faced by consumers of vocabularies is often not solved by merely the vocabularies themselves. Several types of problems surround the use of subsets or supersets of the vocabularies. An intelligence analyst focused on Middle Eastern developments may need precise and accurate coding for only a few of the country codes listed in ISO 3166. A subset including the only the codes they use daily can suffice and the rest of the codes in the list are noise. In stark contrast, organizations like the U.S. Navy can leverage all the country codes ISO 3166 can provide but could benefit from extensions to the list that cover bodies of water, and subdivisions thereof. Either group might copy the values they need from the original country code list, but would then need a mechanism and workflow to update their copy should the original ISO 3166 list get updated. Further, if copies of the code list are modified, then how can future authoritative updates get folded in without collisions between the idiosyncratic modifications and future changes? A version control system with an easy-to-use editor would help, as would the ability to select only portions of vocabularies and recombine them using Semantic Web technology.

A further challenge involves authoritative mapping between vocabularies. Situations arise where a significant amount of legacy information coded with one vocabulary such as FIPS10-4 needs to be integrated or interoperable with information coded with similar yet different vocabularies for the same domain such as ISO3166-2. Whereas there is an authoritative source for each of the respective vocabularies, it is difficult to say who might be the correct source for the mapping between vocabularies. While most users versed in geography can agree that Myanmar is indeed Burma, in other lesser known domains this can pose a difficulty that absolutely requires domain expertise. Mappings can be created but need to be maintained and updated as vocabularies change over time. The Semantic Web is an ideal solution for this problem. Yet there exists no Semantic Web-based vocabulary infrastructure capable of meeting this need.

3 The Vocabutek Solution

We have begun development on a soliution this problem, which we call *Vocabutek*. Vocabutek attempts to solve the above-mentioned shortcomings of vocabulary discovery, creation, hosting, and pricing by deploying Semantic Web technologies in combination with an auction approach based on consumer-driven constraints to determine the monetary cost of a vocabulary. By consumer-driven constraints we mean the selection of attributes of the vocabulary by an consumer, such as the desired upper limit of the monetary price, the data formats, the number of nodes and relationships in the vocabulary, quality assurance, and the like. The cost of a vocabulary is any real-valued number or unit of exchange that can be assigned to a vocabulary in order for there to be a transaction of the vocabulary from the consumer to the provider,and in the case of Vocabutek is usually monetary price.

Vocabutek uses auction mechanism between the consumer in combination with a match of consumer-driven constraints to determine a monetary price for the vocabulary. This can be considered similar to using a market like *eBay* to discover the price of vocabularies. The key insight is that deploying auction mechanisms in a decentralized environment can help provide both an optimal vocabulary and an optimal price for the vocabulary. By matching or causing the production of the vocabulary with the constraints given by the consumer, and then using these to determine price, the service helps insure that the consumer will actually use and, if needed, purchase the vocabulary, while this furthermore makes it easy for the provider to align

[3] *http://neologism.deri.ie/*
[4] *http://rdf.data-vocabulary.org/rdf.xml*

their vocabulary with actual information needs and to exchange their vocabulary with as many consumers as possible. The feedback produced by this type of alignment can drive efforts to solve the challenges facing consumers and providers.

Vocabutek also provides a number of 'Web 2.0' features to increase the likelihood of a successful transaction for both the consumer and the provider. First, Vocabutek can be considered a community portal and social networking site for knowledge representation users and professionals. We provide personal profiles detailing expertise for provider and consumers respectively that allow consumers and providers to discover each other. Secondly, after each transaction, reviews and ratings systems allow providers and consumers to determine the reliability, expertise, and other qualities that may lead to a successful transaction. Producers, seeking to improve their ratings, are encouraged by the feedback given by consumers to improve the marketability of their offering.

While currently the front-end of Vocabutek is based on Drupal, the actual function of the service is a Sesame back-end consisting of an index of vocabularies, where the vocabularies are stored natively in a Sesame triplestore. The matching is performed via SPARQL queries generated the constraint-based mechanism as well as information retrieval over the raw text of the vocabularies. Semantic Web technology is crucial to allow vocabularies to be kept distinct and inter-operate. We convert each term in the vocabulary, regardless of its original form (often XML-based on using some text-based format), to a Semantic Web term and assign it a unique URI. For XML-based vocabularies, we simply concatenate the namespace URI with (if needed) a fragment identifier and the local name, while for text-based vocabularies we generate our own URI identifiers for each term. This lets us represent relationships between components in distinctly separate vocabularies to facilitate interoperability between vocabularies through use of terms like *skos:broader* and even the infamous *owl:sameAs*. This architecture for supporting crosswalks between vocabularies strengthens each and every URI.

4 Detailed Description of Vocabutek

In this section, we explore the functionality of Vocabutek in a step-by-step manner. Vocabutek users can assume two distinct roles, consumers and providers, and an auction mechanism unites them. These roles are thoroughly visualized in the flowchart given in Figure 2, where for purposes of space, vocabularies are called 'KR' for 'knowledge representation.' In the following text, we step through the typical interactions with Vocabutek as given in the flowchart. When a user attempts to access the service, if the user has never used the service before, then their personal details are gathered and a secure OpenID-compatible login is given. In order to facilitate successful transactions, much like on *eBay.com*, the identity of the users can be hidden to some extent as Vocabutek takes care of any financial details of the transaction, although of course it is impossible for Vocabutek to completely hide anyone's identity. This does lead to the possibility that users will contact each other using Vocabutek and then arrange the financial transaction for the vocabulary off-line. However, we believe our easy-to-use services for finding, creating, updating, buying, and selling vocabularies, as well as our value-added services for vocabulary maintenance and URI persistence will provide enough added benefit for users to keep coming back to Vocabutek. Much like a social-networking site, each user maintains both contacts as well as a list of requests for vocabularies and vocabularies they maintain, as well as updates from their co services to build a community of consumers and providers, such as a calendar for upcoming events for the vocabulary community and a library of related online articles and tutorials on creating vocabularies. Upon logging on, the user is given the chance to either request a vocabulary be created, and so instantiates consumer role, or create and modify a vocabulary, and thus instantiate a provider. Note that users may assume either one of these roles using the systems at different times. We first will explicate the role of the consumer.

4.1 Consumers

When a consumer has visited the service before, the consumer can check if any of their information needs for vocabularies have been satisfied by providers using the service. A consumer may also update their personal details at any time once they have logged-in, and browse through the vocabularies hosted by Vocabutek, as shown in the screenshot given by Figure 1. Most importantly, once they have logged in, the consumer then can formulate their information need to the service in for current or future providers to be aware of their need. First, the consumer completes a form describing their information need in a request for a vocabulary. This form can provide a number of constraints over attributes each of which can have multiple values. Some of these values may range over free natural language text, while others may be range over integers, real numbers, dates, relationship types, provenance, or some other ordinal or nominal value. While some of the attributes, and thus constraints, will usually be shared between various vocabulary requests, such as a free text description of the information need, these multi-valued attributes can not necessarily be predicted by the service and listed beforehand, and so a user may enter a new kind of attribute in a structured manner using a form that allows them to create new kinds of constraints to the vocabulary. In order to expedite the user and to allow easier matching of the constraints, a number of templates are given to the provider, who then selects what the name, description, and value ranges of the attribute are.

For example, some of the default kinds of constraints that are provided by Vocabutek are ranges over attribute values are preferred maximum price of vocabulary, size of vocabulary, description of the information need in natural language, which natural languages the documentation of the vocabulary is provided in, obligatory terms needed to be in the vocabulary, dates vocabulary is needed by, subject of the vocabulary, and the formal languages that express the vocabulary, ranging from simple RDF Schemas to non-Semantic Web formats such as KIF (Knowledge Interchange Format). These constraints are built using a number of HTML forms, ranging from a free text box for a natural language description of the information need to be provided, while a drop-down menu with possible categories that describe the information need, calendar that describes the date the vocabulary is needed by, an drop-down box that allows popular natural languages the vocabulary should be provided to be chosen, and a radio-button that specifies which popular vocabulary format that the vocabulary is wanted in. Importantly, a preferred price range for the vocabulary may also be given. In order to maximize the likelihood that the provider's vocabularies ranges over some of the same attributes as the consumer's vocabulary request does, the service suggests templates for possible attributes in order of popularity of those used by previous users, although those forms used by other consumers of vocabularies in the same category are provided. In this way, the matching process between vocabulary request and vocabularies stored by the service are more likely to match, as it is easier to automatically match known attributes with constraints rather than do indexing and searching over novel constraints. The vocabulary re-

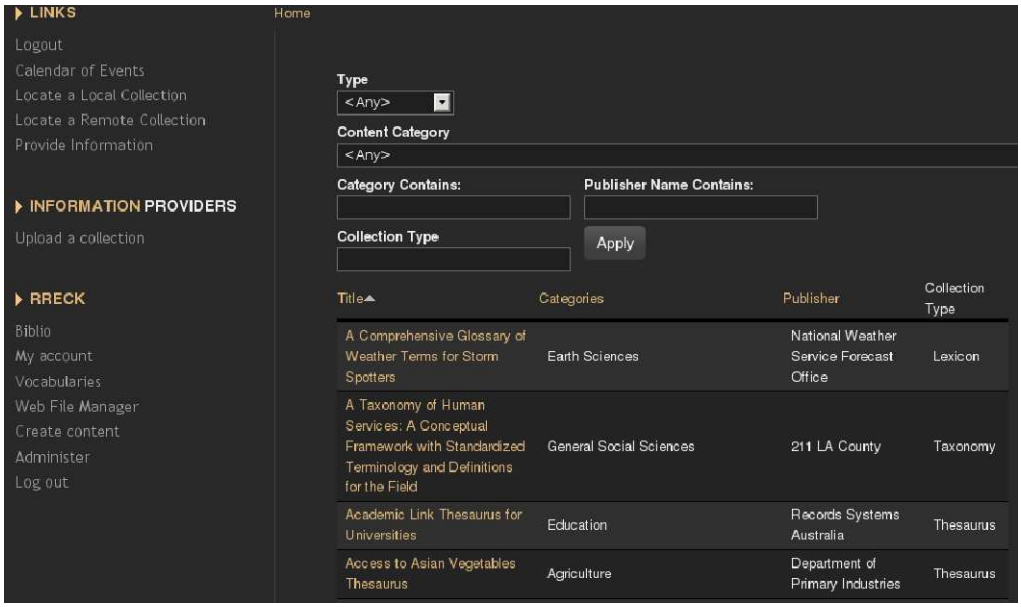

Figure 1. Browsing and Searching Vocabutek

quest is then recorded in the database. This information need, now phrased as a request for a vocabulary using the constraints provided by Vocabutek, is then publicly available to other users of the sites, including providers. Now we move to describing the role of Vocabulary providers.

4.2 Vocabulary Providers

When specifying their role as an provider, the personal details are gathered as is done for consumers, but in addition information about their expertise is gathered via a series of free-text and multi-valued attributes, including allowing the provider to provide via check-box their expertise in a taxonomy of vocabulary types. Once registered as an provider, they may search and browse the various requests for vocabularies. The information concerning their expertise is gathered by the system and can also then be used to alert the provider for any possible relevant vocabulary requests they can fulfill. A provider may register a vocabulary with the service at any time. This may include any number of vocabularies that provider has created regardless of their possible applicability to any particular vocabulary request on Vocabutek. There are, in general, two kinds of vocabularies on Vocabutek. The first is those that are directly stored on Vocabutek, and the other is the kind that has simply been reference by Vocabutek, but is hosted elsewhere. There are two other options in between hosting and storing the vocabulary completely independently. The first one one in which Vocabutek provides a redirect service for persistent URIs for a vocabulary already existing elsewhere, and the second where Vocabutek provides documentation and hosting services for a vocabulary whose URIs are already in a domain name owned elsewhere. Regardless, there is an advantage to registering a vocabulary with Vocabutek, as it allows this vocabulary to be discovered by consumers using Vocabutek even if the vocabulary is hosted elsewhere.

In any case, the provider can fill out a form describing the attributes of their vocabulary and post this vocabulary description to be available to other consumers. As the vocabulary is usually digital, the vocabulary may be uploaded directly to the service and then the attributes can be partially derived and recorded by the system in a database of vocabularies. For example, the system may automatically record the number of items in the vocabulary, what formats the vocabulary is in, the properties of the relationships between terms (i.e. associative/hierarchical), and the time of upload to the service of the vocabulary. Some attributes of the vocabulary may not be automatically derived from the vocabulary. In particular, the estimated selling price of the vocabulary may be initiated by the provider. The provider also enters information such as their intended maintenance of the vocabulary, its original reason for production, and the other pertinent information, including social and contractual information. Note some vocabularies that are available without any charge (and so are available free of cost, even though payment may be required for maintenance or changes), the entire vocabulary may be available using Vocabutek, including being being exposed to the Web for normal hypertext search engines to locate. Otherwise, only a limited part or none of the vocabulary may be publicly available for consumers of Vocabutek and will not be indexed by search engines, as total access to the vocabulary may require a financial transaction.

Furthermore, Vocabutek provides a Web-based interface for the modification and creation of vocabularies, as many users may not have their own custom-made tools for creating new vocabularies, and it is in the best interest of satisfying requests to allow as many people as possible to participate. These providers may possess vocabularies that they have gathered from other sources or have them in some non-digital medium, but lack the ability to easily modify them and provide them in formats for use by other users. Thus, one capability of our service is a mechanism that may be used online for creating and modifying vocabularies. The interface allows the provider to easily list a number of terms they are aware of. This interface also supports automated or semi-automated extraction of such terms from various sources of information, such as the extraction of high-information value terms and named entities from documents. Once a potential list of terms is created, the interface allows these vocabulary items

to be related systematically using a number of terms from SKOS, OWL, and RDF(S), as determined by our empirical analysis of vocabularies given earlier [2]. Whether or not the lists of vocabulary items is ordered (and thus an *rdf:List*) can be provided. Annotations, such as comments via *rdf:comment* and natural language labels via *rdf:label* all be added to the terms, and the option to do in multiple natural languages is given, and then the language types are tracked in the creation of the literal values of the vocabularies. The interface then allows new terms to be added dynamically as more structure is added to the initial list. If the terms are from RDF(S) or OWL, the interface can automate this process of inference by using Pellet on the back-end and provide the results to the provider in order for them to be better understand and possibly modify their vocabulary. As the process is iterated by the user over time, a list of terms, which by itself is a very simple hierarchical vocabulary, can be transformed by the provider into a more structured vocabulary. Vocabutek provides revision-based changed control, comments, and a discussion forum to encourage communication. In particular, these facilities allow multiple providers to collaborate over time, and we imagine that often collaborations will develop between domain experts and knowledge representation engineers. We also allow providers to pass the control and maintenance of a vocabulary to other providers. The provider may even sell or relinquish the rights to further control and delegate a vocabulary to another provider. The provider can, via a form, establish a legal contract with Vocabutek for maintenance of vocabulary. We allow a range of options, ranging from a vocabulary that is free to distribute and modify with royalty-free status to vocabularies that may not be distributed or sold without the service, or even pass legal ownership of the vocabulary to the service, perhaps for some pre-determined time.

4.3 Auctions for Determining Vocabulary Cost

Our system deploys the multi-valued constraint matching algorithm that generates SPARQL requests from the constraints given by requests and the existing vocabularies in the index of Vocabutek. The constraints of the consumer are not usually totally satisfied by one or more vocabulary, but only partially satisfied by any number of vocabulary indexed by the service. For example, often one constraint of the vocabulary is that it is has a coverage of certain terms and therefore has a certain minimum size. However, very rarely will a pre-existing vocabulary cover all the terms needed by a consumer. Furthermore, the vocabulary will usually have many terms or attributes not needed by an consumer. So, the constraint of the minimum number of terms can only be partially matched to a number of vocabularies, which often feature too many terms. The constraints given by the user then partially match a potentially large number of vocabularies. Vocabutek retrieves from its index each of the vocabularies that match at least one of the constraints given by the consumer. Then, ranking the vocabularies by the weighted number of constraints matched, where the constraints are weighted in the order they are given by the user, the vocabularies are presented to the consumer based on a ranking determined by the number of constraints that are matched, and for data-ranking, LuceneSail is used over the Sesame triple-store.[5] The consumer can then inspect each vocabulary that potentially matches their constraints in order to discover the best match for their information need. Furthermore, its possible that the consumer may not be satisfied with any of the vocabularies for whatever reason, so the request may be fulfilled at any time by custom-made vocabularies, and providers may attempt to modify an exiting representation so it better fits the current vocabulary request.

Once one or more vocabulary that potentially matches the constraints are retrieved, a bidding process between the consumer and the providers commences that determines the monetary cost of the vocabulary. This process may be automated, semi-automated, or non-automated. In an automated version, if multiple vocabularies match the users constraints, then the entire process of finding the best vocabulary for the consumer can be considered a multi-attribute auction, and an algorithm that optimizes the matching constraints while determining a new cost may be automatically determined using the algorithm explored in Koppius and van Heck [3]. In a fully automated auction, if one of the constraints of the consumer is a minimum cost for providing the vocabulary, then if the algorithm provides a cost below the minimum cost, the transaction that provides the consumer with the vocabulary from the provider can be fully automated. However, if an automated algorithm to determine the cost of the vocabulary is not provided, or if the cost determined by the algorithm does not fall below whatever minimum cost could be provided by the provider, then a non-automated process of bidding begins. The consumer can then use the service to contact one or more the provider and communicate with them over the precise constraints of the vocabulary, asking them to modify the vocabulary itself and change the associated attributes of the vocabulary, including the associated cost. This process can then be iterated until the consumer's constraints are satisfied, or the consumer decides to terminate the process. If a vocabulary is accepted, then the service automatically transfers the vocabulary from a provider to the consumer, and then the consumer owes the cost of the vocabulary to a provider. If the vocabulary is given its cost in some monetary value, then monetary cost of the vocabulary is securely transferred from the consumer to the provider. If the entire vocabulary is not available without a monetary transaction, then the vocabulary is then made available to the consumer after the transaction.

After the transaction, the consumer may then use a number of facilities provided by the service to provide feedback about the vocabulary and their interactions with the provider to the system, and most importantly, to directly edit the vocabulary itself. This feedback can be either metadata about a particular vocabulary produced by an provider, a provider itself, or alterations to the vocabulary itself. Likewise, a provider may provide metadata and feedback about their interactions with an consumer, and update the vocabulary in request to the consumer's needs after the transactions (vocabulary maintenance). This feedback may take the form of a numerical rating, a natural language comment, and other information that may be pertinent to those interested in future transactions with the consumer. Some of this feedback information, such as number of completely successfully transactions and the number of vocabularies they have created, can be automatically gathered by the service and displayed to potential consumers, and statistics taken gathered from the feedback of consumers. Furthermore, some of the comments and ratings may be applied not to providers themselves, but their specific vocabularies they produce, and providers may document feedback about the behavior of an consumer. In this way, if problems or benefits are encountered in the vocabulary in the course of its actual use after the time of the actual transaction, then these are recorded by Vocabutek. The rating of both a producer and a vocabulary can be taken into account in the ranking of vocabularies for a consumer in response to a request.

[5] *http://dev.nepomuk.semanticdesktop.org/wiki/LuceneSail*

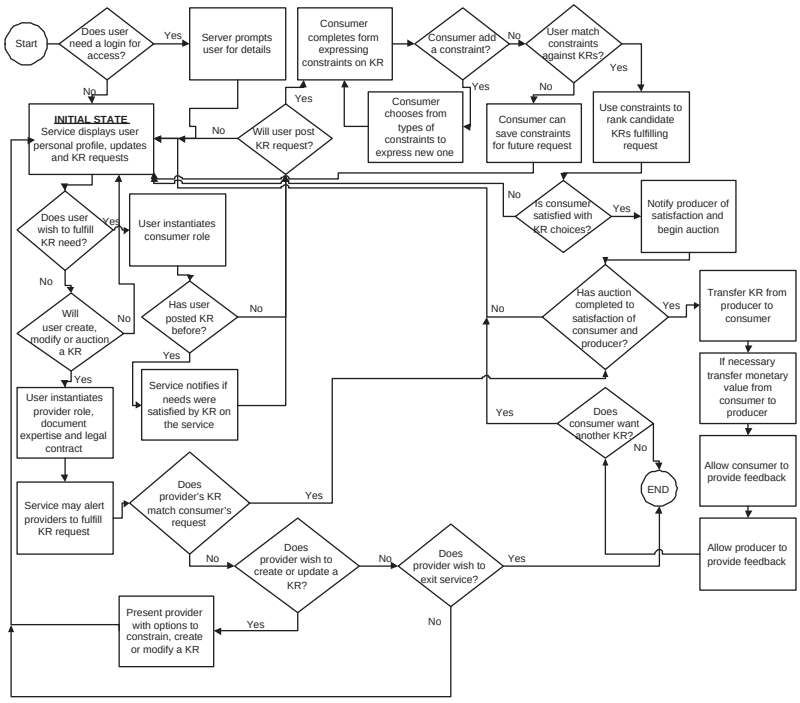

Figure 2. Flowchart of Vocabutek

5 Conclusion

There is of course much work to be done in the future, and work on Vocabutek has only just started. Vocabutek does not solve the entire 'vocabulary crisis' on the Semantic Web, and there still needs to be at least one non-profit, and ideally a few, large general purpose hubs of free vocabularies that can meet the information needs of many people. However, for information needs that are of commercial or specialist interest, Vocabutek provides a way for these users to communicate their interest to those who create Semantic Web vocabularies. First, it provides a simple yet extensible form-based constraint mechanism to specify needs for vocabularies. Then, we provide the ability for providers of knowledge to upload or notify the system of their vocabularies, and to produce vocabularies in response to user-needs, either with their own tools or using our form-based system. Finally, we provide an incentive by allowing users to charge monetary costs, as determined by an auction, for their vocabularies and their maintenance. . The next step is to create a community of users around Vocabutek and to empirically measure how they create and maintain vocabularies in order to determine if the auction-based methodology of Vocabutek works: Can we bootstrap the Semantic Web by bringing market forces to bear on the problem of finding and consuming information? Given the success of combining market-based forces with mass participation in general, this may very well be possible.

REFERENCES

[1] Gong Cheng, Weiyi Ge, and Yuzhong Qu. FALCONS: Searching and browsing entities on the Semantic Web. In *Proceedings of the World Wide Web Conference*, 2008.

[2] Harry Halpin. Is there anything worth finding on Semantic Web? In *Proceedings of the World Wide Web Conference*, 2009.

[3] Otto Koppius and Eric van Heck Information architecture and electronic market performance in multidimensional auctions Erasmus Research Institute of Management. Rotterdam, Erasmus, 2002

[4] Holger Knublauch, Ray Fergerson, Natalya Noy, and Mark Musen. The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications In *Proceedings of the International Semantic Web Converence*, 2004, pp. 229-243.

[5] Elena Simperl, Igor Popov, and Tobias Brger ONTOCOM Revisited: Towards Accurate Cost Predictions for Ontology Development Projects In *Proceedings of the European Semantic Web Conference*, 2009.

[6] Katharina Siorpaes and Martin Hepp OntoGame: Weaving the Semantic Web by Online Gaming In *Proceedings of the European Semantic Web Conference*, 2008.

Diamond: A SPARQL Query Engine, for Linked Data Based on the Rete Match

Daniel P. Miranker, Rodolfo K. Depena, Hyunjoon Jung, Juan F. Sequeda and Carlos Reyna [1]

Abstract. This paper describes a system, Diamond, which uses the Rete Match algorithm to evaluate SPARQL queries on distributed RDF data in the Linked Data model. In the Linked Data model, as a query is being evaluated, additional linked data can be identified as additional data to be evaluated by the query; the process may repeat indefinitely. Casting Linked Data query evaluation as a cyclic behavior enables making a constructive analogy with the behavior of a forward-chaining AI production system. The Rete match algorithm is the most commonly used implementation technique for AI production systems. Where AI production systems are known to make a relatively consistent number of changes to working memory per cycle, dereferencing URIs in the linked data model is a potentially volatile process. The paper provides an overview of Diamonds architectural elements that concern integrating the Rete match with the crawling of Linked Data and provides an overview of a set of Rete operators needed to implement SPARQL.

1 INTRODUCTION

Linked Data defines a set of best practices in order to treat data as a distributed interconnected graph just as the Web, through hyperlinks, has enabled documents to be interconnected and distributed [9]. In the Linked Data model directed, labeled graph edges, known as triples, are defined using RDF, the Resource Description Framework; a triple is comprised of a subject, a predicate and an object. Each component of a triple may be represented by a URI. By definition, the URI will be associated with an Internet server. The Linked Data principles stipulate that when a URI is dereferenced, the server should return a set of triples [2]. Those triples, in turn, may contain URIs for different servers. Thus, there is a potential for a triple on one server to logically connect one to three graph-edges, such that additional graph structured data may be gathered from distributed servers.

SPARQL is a query language for RDF graphs, and is itself commonly described as a language for describing graph patterns [12, 13]. Hartig et al. present an approach to execute SPARQL queries over Linked Data called *Link Traversal Based Query Execution* [7, 8]. In this approach, if a triple satisfies just one clause in a SPARQL query, then the connected components of that triple, linked by URI, may satisfy other clauses. Thus, in the course of evaluating a SPARQL query, if a triple satisfies a clause, each of its embedded URIs must be dereferenced. In other words, for each such URI, it may be necessary to go to a server and collect an additional set of triples.

Observe that a consequence of evaluating a SPARQL query over Linked Data may result in additional data being collected from over the Internet, and further evaluating the SPARQL query in a context that includes this new data. This can be viewed as an iterative process that may continue indefinitely [5, 6, 7, 8].

We observe a similarity in the execution of SPARQL queries over Linked Data queries and forward-chaining rule systems in Artificial Intelligence. In the latter a set of rule antecedents are evaluated against a repository of working memory. A satisfied rule is selected and its consequent executed. The consequent may insert or delete additional working memory elements. The rule sets antecedents are reevaluated. The cycle may proceed indefinitely. Thus, in the Linked Data model, dereferencing a URI and the additional triples fetched can be seen, operationally, as the same as firing a rule that adds elements to working memory.

Forgy's Rete match is the defacto standard for implementing forward-chaining rule systems[3]. Rather than reevaluating the rule antecedents at each cycle, the Rete Match processes incremental changes to the working memory as incremental changes to the set of satisfied rules. This is accomplished by interposing state operators, or memory nodes, in between a network of filtering operators. Incremental changes to working memory are processed as cascading incremental changes to the memory nodes. See Section 3.

2 DIAMOND ARCHITECTURE

The Diamond architecture is illustrated in Figure 1. The Rete network is created, on demand, as queries are entered by a user. The URI dereferencing object is static and not at all complicated. The most critical architectural component is the pair of queues, one for triples, and one for URIs, that connect the Rete network object with the URI dereferencing object, and their manager. These queues are intended to form the basis of parallel, asynchronous, execution of query evaluation and URI dereferencing.

Correctness of the Rete match algorithm requires a change initiated at the top of the network to be processed, depth-first to completion before beginning to process another change. Early in the execution of a query, little data will have been consumed, the network will be nearly empty, and processing will be fast. As data accumulates, one can anticipate query processing to slow, just as a query on a large database will take longer than the same query on a smaller database. Dereferencing a URI may yield an arbitrary number of additional triples. Some of those triples may not match any basic triple patterns in the query and the Rete network will dispatch them quickly. Other triples may propagate through the entire Rete network, and yield new solutions to the query. Even the number of solutions so produced is unpredictable. Thus, the queues act both as buffers, and the synchronization mechanism between two processes whose execution behavior is anticipated to be volatile.

[1] Department of Computer Science, University of Texas at Austin, email: {miranker, jsequeda}@cs.utexas.edu, email: {rudy.depena, polaris79, creynam89}@gmail.com

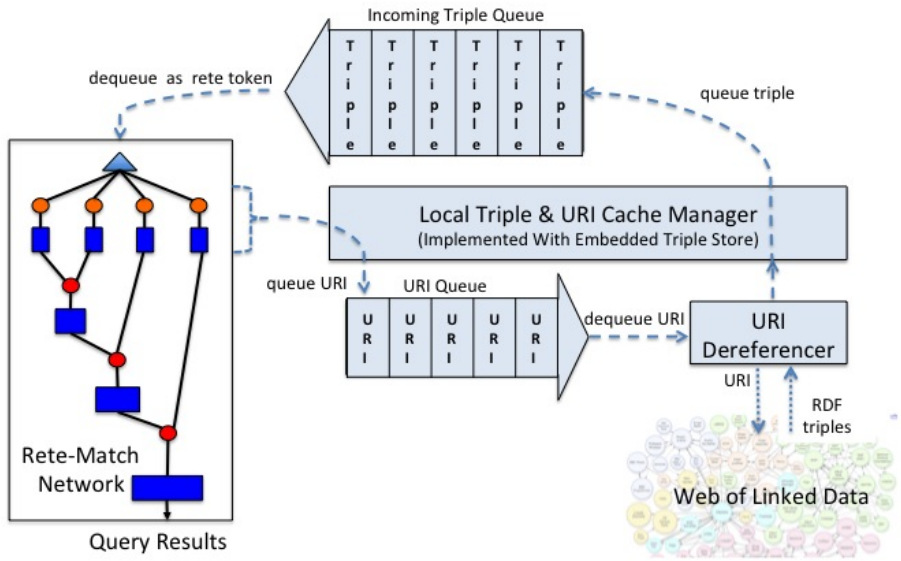

Figure 1. Diamond Architecture Diagram

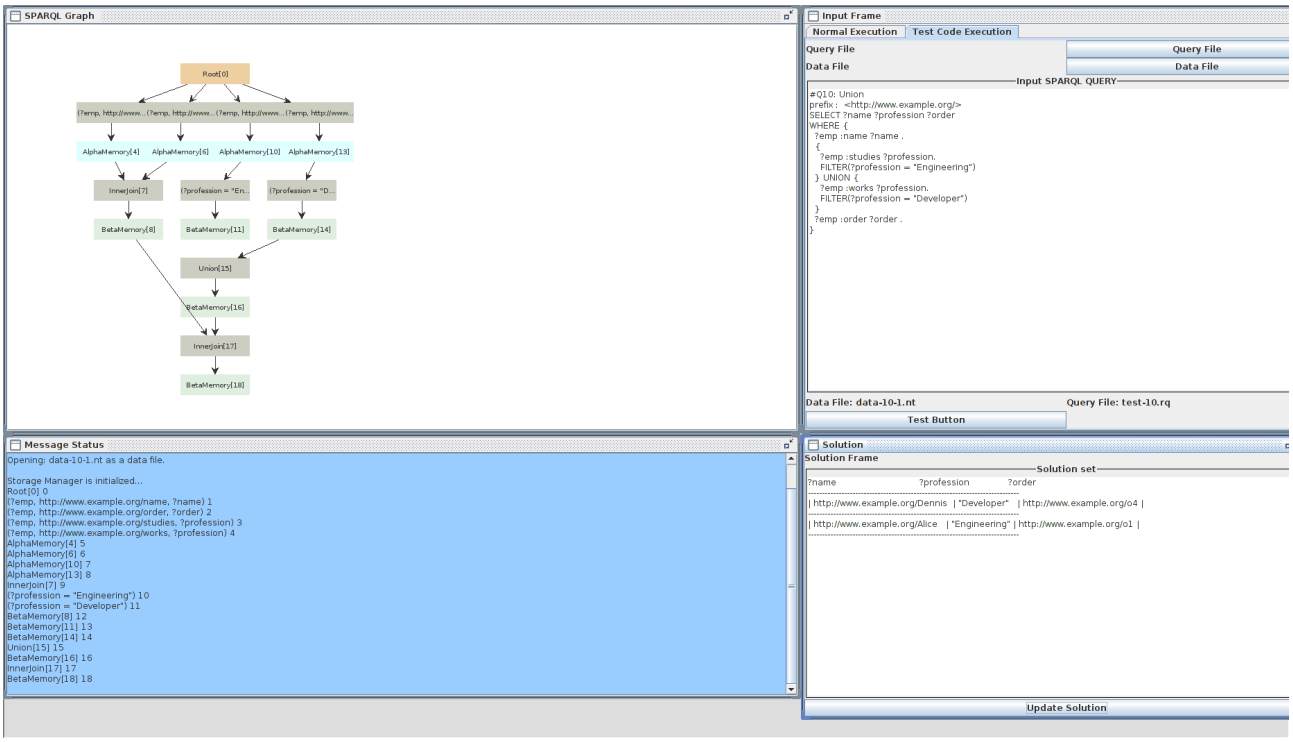

Figure 2. SPARQL rule graphical debugger

The function, initially, of the triple & URI manager is to avoid redundant dereferencing of a URI and, similarly, redundant processing of triples through the Rete network. The manager is currently implemented using the Sesame[2] triplestore. The intrinsic depth-first processing requirement of the Rete match coupled with the intuition behind the Link Traversal model suggests treating the dereferencing queue as a stack (LIFO) is a natural, good fit. Or, the triple queue may be treated as a stack. If priority is given to both the most recent URI, and the ensuing triples, it is plausible that processing will focus strictly on the first few triple patterns of the query. It is easy to construct adversarial graphs such that no progress can be made in completing the full pattern of the query until traversals satisfying the first few triple patterns are exhausted.

As the research moves forward we may consider the use of priority queues ordered by means of heuristics concerning provenance or develop optimization strategies driven by statistical models based on historical behavior. In other words we may bias the system to fetch data from higher quality and/or historically faster data sources. The formal semantics of Linked Data queries is still being determined [6]. It is possible that open issues may be resolved in ways that limit operational behaviors. For example, if fairness properties are stipulated, deterministic methods that favor certain servers may induce starvation of other servers.

Implementing the triple & URI manager by using an embedded triple store adds flexibility per two additional open issues, one concerns speed, the other semantics. Motivated, in part, by *Big Data* computing, some linked data systems compute queries against a large local cache of precrawled data [11]. By committing, at the onset, to embedding a triple store in the cache manager, the evaluation of this approach can be compared to a strict crawling of live data while minimizing additional engineering and thereby maximizing the control of the experiments. The semantic issue concerns the Link Traversal Based Query Execution Model. Consistent with thath model, our first implementation only dereferences URIs of triples that have passed the initial Rete-Match filters and are recorded in a memory node. However, we promptly observed the following. If query solutions are subgraphs of disconnected graphs, following an initial set of links may yield only a subset of the possible solutions.

The system includes a SPARQL rule debugger typical of graphical debuggers in Rete-based rule execution systems shown in Figure 2.

3 RETE NETWORK

A Rete network is comprised of filter nodes, interleaved with memory nodes. To implement SPARQL using a Rete network is to map the pattern testing primitives of SPARQL to Rete operators. Beginning with the TREAT algorithm the connection between Rete operators and database operators has been exploited by many [10, 4] and we will do so here. First we illustrate, through an example, that pattern matching in SPARQL differs little from early, Rete-based AI rule languages, and then detail the corresponding Rete network and its behavior with respect to incremental addition of data; See Figures 3, 4, 5.

It is convenient to think of memory nodes as containing the solution to a subset of the query (rule antecedent). Memory nodes that store the output of a unary operator are called alpha memories. Memory nodes that store the output of binary operators are called beta memories. The last beta memory contains solutions to the query.

We explain the contents and a change of state for the example Rete network. Initialization for a particular query starts with dereferencing the URIs in the query. The <http://cs.utexas.edu/miranker> URI is dereferenced and a set of RDF triples are returned. One of those triples includes (<http://cs.utexas.edu/miranker> :knows <http://web.ing.puc.cl/arenas>). As illustrated in Figure 4, a Rete network is found in a state having processed some sample triples. The existence of the "Arenas URI" in the first alpha-memory suggests additional information may be found at http://web.ing.puc.cl server. For simplicity assume the "Arenas URI" is dereferenced and the return result includes the triple (<http://web.ing.puc.cl/arenas> :age "28"). This triple now satisfies the SPARQL graph pattern. Figure 5 illustrates the state of the network after processing this new triple that matches the one-input node on the right side and satisfies the condition for the two-input, or *Join* node. A quanta of information that moves through Rete network is commonly called a token. Figure 5. Although our implementation is in Java, one can think of an initial token being a pointer to the new triple, entering the root of the network, distributed across the two filter operators, passing the "age" filter. This success is recorded in the alpha memory. The token proceed to a join node which may identify a consistent binding for variable ?x. Each binding is emitted from the join node as a new token. A copy of the token recorded in a beta memory.

```
SELECT ?age
WHERE {
  <http://cs.utexas.edu/miranker> :knows ?x .
  ?x :age ?age.
}

(P example-rule
  (http://cs.utexas.edu/miranker :knows <x>)
      (<x> :age <age>)
-->
     (make <age>)
```

Figure 3. A SPARQL Query and Its Equivalent in OPS5 Syntax

A Rete operator set for SPARQL follows from work by Angles and Guitierrez concerning the semantics of SPARQL [1]. Angles and Gutierrez proved that SPARQL is equivalent in expressive power to non-recursive Datalog with negation. Given the relationship between SPARQL, Datalog and relational algebra, to form a Rete operator set for SPARQL one needs to identify a mapping, by transitivity, from SPARQL syntax and its native logic, to relational operators. Table 1 summarizes the preceding construction including the relationship between SPARQL constructs, SPARQL algebra, formal semantics, and Rete operators.

Each Rete-Match operator is implemented as a unique node in the network, with its object-oriented structure represented in Figure 5.

3.1 Root

The root node is the gateway for all data that enters the network. When a working memory change enters the network in the form of a tuple, it is wrapped in a structure called a token which includes a tag indicating whether the element is to be added (+ tag) or deleted (- tag) from memory. The root node then propagates the token to its direct children, all of them TriplePatternTest nodes.

[2] http://www.openrdf.org/

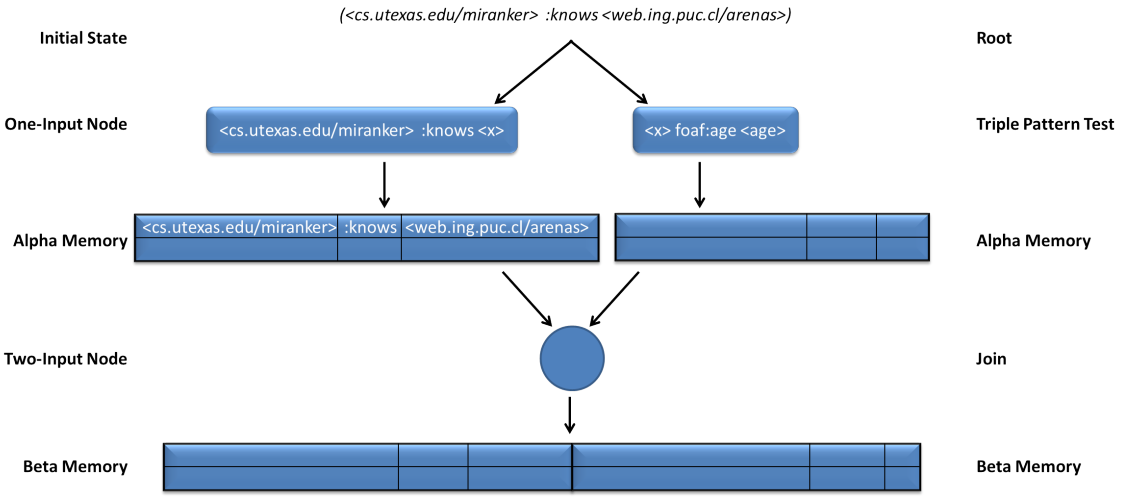

Figure 4. Rete network designed by Diamond for the SPARQL query from Figure 3; the memories have been previously filled with sample data.

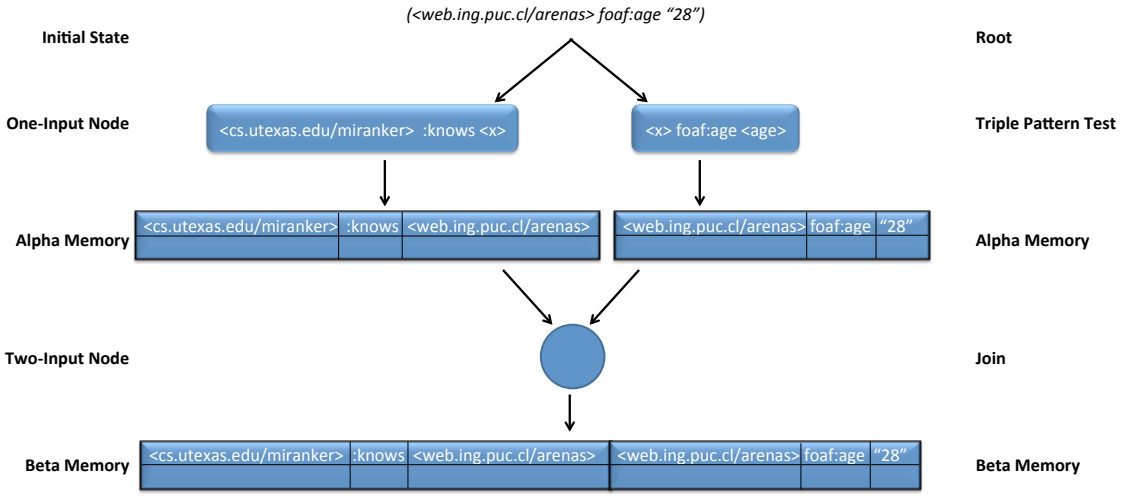

Figure 5. State of the Rete network after introducing the new triple. Labels on the left correspond to the terminology used in the original Rete match algorithm while labels on the right correspond to the names used on Diamond to convey the semantics of SPARQL.

Table 1. Summary of the Derivation of SPARQL Rete Operators

SPARQL Language Construct	SPARQL Algebra Operators	SPARQL Operators in [1]	Rete Operators
BGP	eval(D(G), BGP)	T(GroupGP)	TriplePatternTest(tp, R)
(.)	JOIN(Ω_1, Ω_2)	P1 AND P2	R1 InnerJoin R2
OPTIONAL	LEFTJOIN(Ω_1, Ω_2, C)	P1 OPT P2	R1 LeftJoin R2
UNION	UNION(Ω_1, Ω_2)	P1 UNION P2	R1 Union R2
FILTER	FILTER(C, Ω)	P1 FILTER C	Select(C,R)
SELECT	PROJECT(Ψ, PV)	SELECT	Project(s,R)

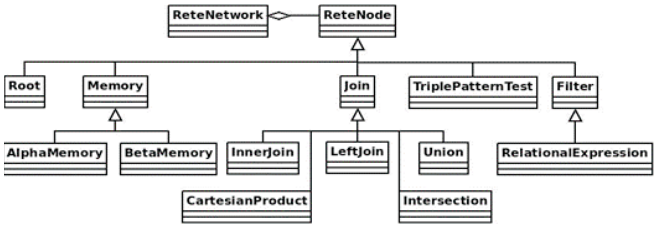

Figure 6. UML Class Diagram for the Rete-Mach implementation

3.2 TriplePatternTest

A unary test node that propagates only those tokens whose RDF tuple matches the constants in a single triple pattern; matching tokens are sent to the subsequent child alpha memory.

3.3 Memory

Alpha and Beta memories store the contents of + tokens they receive and delete tuples matching the contents of the - tokens.

3.4 Inner Join

All Join operators have two memories as their parents. The inner join operator is activated by receiving a token from either the left or right side.

The token will be compared with every token contained within the opposing memory, and two tokens are joined together if their variable bindings are consistent or no shared variables exist in either token. The resulting joined token will then be propagated to the succeeding Beta memory for storage.

3.5 Left Join

Behaves like an Inner Join, but with a couple of special cases:

If a token from the left parent does not match a token from the right hand side, then the left hand side token will propagate to the child Beta memory without being joined; else the tokens are joined as usual.

If a token comes from the right hand side parent, it is compared with the left side, and for each match a token with the joined tuples is propagated alongside with a negated token containing the matching left side tuple.

3.6 Union

A binary operator that computes the set union between the two memories, it propagates all + tokens that are not already contained in the succeeding child memory and only those - tokens that match a tuple in that same child memory.

3.7 Intersection

A binary operator that computes the set intersection between the two memories. It propagates only those + tokens that match a tuple in the opposing side memory and are not already contained in the succeeding child memory. It also propagates only those - tokens that match a tuple in the succeeding child memory.

3.8 Cartesian Product

Whenever a token reaches one side of this binary node, a new token with the same tag is created and propagated to the subsequent child Beta memory for each element of the other side memory containing the joined tuples.

3.9 Filter

Compares the contents of each + and - tokens to a local constraint, if there is a match the token will propagate to the subsequent Beta memory, else it stops.

4 STATUS and PRELIMINARY RESULTS

To minimize the learning curve of future project participants Diamond is constructed using canonical Java compiler tools. The lexer and parser are build using a BNF grammar description of SPARQL and JavaCC4. Java Tree Builder6 (JTB) is the basis of internal syntax tree. At runtime, a SPARQL query is parsed into internal form, and the Rete network created by traversing the syntax tree and calling the Rete network object constructors.

The system is not yet multithreaded. In other words there is no parallelism among the query processing and the URI dereferencing objects. The two queues are both implemented as standard FIFO queues.

The system successfully passes all 126 SELECT queries in the RDF API for PHP test suite (RAP). The suite contains another 25 test cases consisting of SPARQL, ASK, DESCRIBE, CONSTRUCT and solution modifier queries, which are currently not supported in Diamond.

For our preliminary experiments, we used the Berlin SPARQL Benchmark. We execute every experiment ten times (excluding three warm-up runs) and calculate the average query execution speed. See Table 2. Although the test data source was hosted on a machine on the local network, without multithreading, the execution times include the total network latency for dereference every URI in the query.

Table 2. Timing Results from Linked Data queries

Query	Query 1	Query 2	Query 3
# of Results	10	1	4
# of Tokens Processed	10,246	10,106	64
Execution Time	5 min, 47 sec	3 min, 43 sec	2.12 sec

5 FUTURE WORK

This paper presents the architecture of Diamond, a SPARQL query engine for Linked Data based on the Rete Match. The system has only just recently become functional. The system architecture anticipates future development and evaluation of optimization with respect to prioritizing dereferencing URIs, the processing of the resulting triples, and local caching of linked data. Future evaluation will necessarily include comparison with other Linked Data query systems, most notably SQUIN [3]. We made observation in several places in this paper that there are open implementation choices still to be researched that may be influenced or even disallowed, depending on how the semantics of these systems are resolved.

ACKNOWLEDGEMENTS

This research was supported by NSF grant IIS-1018554. Juan F. Sequeda was supported by the NSF Graduate Research Fellowship.

REFERENCES

[1] Renzo Angles and Claudio Gutierrez, 'The expressive power of sparql', in *International Semantic Web Conference*, pp. 114–129, (2008).
[2] Tim Berners-Lee. Linked data - design issues. `http://www.w3.org/DesignIssues/LinkedData.html`.
[3] Charles Forgy, 'Rete: A fast algorithm for the many patterns/many objects match problem', *Artif. Intell.*, **19**(1), 17–37, (1982).
[4] Eric N. Hanson, 'The design and implementation of the ariel active database rule system', *IEEE Trans. Knowl. Data Eng.*, **8**(1), 157–172, (1996).
[5] Olaf Hartig, 'Zero-knowledge query planning for an iterator implementation of link traversal based query execution', in *ESWC (1)*, pp. 154–169, (2011).
[6] Olaf Hartig, 'Sparql for a web of linked data: Semantics and computability', in *ESWC*, pp. 8–23, (2012).
[7] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag, 'Executing sparql queries over the web of linked data', in *International Semantic Web Conference*, pp. 293–309, (2009).
[8] Olaf Hartig and Johann-Christoph Freytag, 'Foundations of traversal based query execution over linked data', in *HT*, pp. 43–52, (2012).
[9] Tom Heath and Christian Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers, 2011.
[10] Daniel P. Miranker, 'Treat: A better match algorithm for ai production system matching', in *AAAI*, pp. 42–47, (1987).
[11] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello, 'Sindice.com: a document-oriented lookup index for open linked data', *IJMSO*, **3**(1), 37–52, (2008).
[12] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez, 'Semantics and complexity of SPARQL', *ACM Trans. Database Syst.*, **34**(3), (2009).
[13] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, `http://www.w3.org/TR/rdf-sparql-query/`.

[3] http://squin.org

Implementation of a SPARQL Integrated Recommendation Engine for Linked Data with Hybrid Capabilities

Sean Policarpio and **Sören Brunk** and **Giovanni Tummarello** [1]

Abstract. Linked data can serve its purpose in many real-world applications including recommendation engines. We present here our approach and implementation for a recommendation engine that uses linked data as input for both collaborative filtering and content-based recommendations. The SPARQL query language is integrated for the purpose of allowing users of the system to declaratively specify RDF data for recommendation computation. The system has the ability to create or augment linked data by outputting recommendations in RDF and can also perform hybrid recommendations. It can handle large amounts of linked data input due to the systems parallel programming framework.

1 Introduction

As the amount of data within the linked open data cloud [1, 2] increases, the potential for its use in different applications and purposes grows. One such field of application is in the domain of *recommendation engines*. Recommendation engines (or simply *recommenders*) [6] serve the purpose of producing relevant suggestions for persons and things based on prior knowledge (e.g. user preferences or meta-data). Their use in companies such as Amazon and Netflix have been frequently discussed. However, recommenders designed and actually implemented for use with linked data or "the web of data" have not been widely explored and in our opinion have potential for improvement. We present our implementation of a recommendation engine with a capacity for linked data and the ability to compute two types of recommendations:

- Collaborative filtering [9] is a form of recommendation based on user preferences towards items. Recommendations exist as probable preferences that users may have towards items which they have not expressed before. They are computed by determining similarities between users or between entities based on preexistent user preferences.
- Content-based recommendations [9] on the other hand do not necessarily have to involve information about user preference. Instead, they compute recommendations by directly comparing items based on their *features* or meta-data. For example, items are most similar when they share the most common features. Similarity, indicated in most cases by some numeric value, can be used when suggesting relevant or related items.

[1] Digital Enterprise Research Institute, National University of Ireland Galway, email: firstname.lastname@deri.org

What makes linked data desirable for use with recommendation engines is its heterogeneous structure and ubiquitous nature. A recommendation engine that is capable of interpreting data from various and different sources has the potential to compute diverse suggestions across multiple domains. For example, in terms of the Resource Description Framework (RDF), a recommender could generate content-based recommendations for books (defined in an RDF graph) based on their similarity to movies (located in another RDF graph). This would be possible if the books and movies shared relatable features (e.g. *genre, authors, writers, era*, etc.) that were retrievable via linked data. Recommendations could be refined or improved by selectively including (or excluding) features from either of the linked data sources.

A recommendation engine like this also has the potential to create or augment linked data. Computed recommendations can be transformed into new linked data and can retain links to the original input data by referencing a shared URI. Besides having the recommendation data accessible through RDF, this is advantageous because of the cyclic relationship in which the linked recommendation data can be used to compute or refine further recommendations within the engine. We discuss this relationship in more detail when introducing our recommenders ability to perform *hybrid* recommendations [3].

An aspect of linked data which we believe is both beneficial and well-suited for recommendation engines is the way in which it is usually accessed; through the use of the declarative SPARQL query language. SPARQL queries provide a well structured, expressive, and reliable way for users to define the data that should be used for computing recommendations. For example, in regards to a content-based recommender, a SPARQL query for a fictional art auction house can be written to retrieve auction items like so:

```
SELECT ?artPiece ?artist ?year ?medium ?influence WHERE {
  ?artPiece a :AuctionItem.
  ?artPiece :artistName ?artist.
  ?artPiece :created ?year.
  ?artPiece :medium ?medium.
  ?artist :influencedBy ?influence.
}
```

To perform similarity computation of the results, the data can be used with a content-based algorithm such as co-occurrence or Tanimoto similarity [7].

The challenge in such a recommendation system lies in the framework and design to accommodate both linked data and

recommendation algorithms. The first intuitive approach to integrate the use of linked data in a recommendation engine is through the implementation of a SPARQL interface for specifying and inputting RDF data like that of the previous query example. However, another aspect that must be considered is scalability; to be truly useful, such an engine must have the ability to work with the large volumes of available linked data. In this case, we consider a distributed model (i.e. Hadoop/MapReduce) for recommendation computation. Along with a number of other implementation components, the main contribution of this paper is to present and discuss the approach, design, and construction of a working and practical linked data recommendation engine that supports these considerations.

Before detailing the system framework, in Section 2 we present a formal representation of the recommendation engine. We then discuss the major components of our recommendation engine. Section 3 presents case studies to demonstrate the use and workflow of the system. We define hybrid recommenders for our system in Section 4. Finally, in Section 5 and 6 we discuss related work and conclude with future plans.

2 A Recommendation Engine for Linked Data

Before the construction of the engine, formal definitions of the recommenders were written to aid in the systems design. Our collaborative filtering and content-based recommenders have a common workflow; they involve the retrieval (or extraction) of data, the aggregated computation of recommendations, and finally output. These definitions are based on general recommendation algorithm approaches and have been adapted for RDF input and output.

Collaborative Filtering The collaborative filter defined here is called an item-based recommender [10]. For determining relevant item recommendations, this algorithm computes similarities between items based on the preferences made by other users. A collaborative filter is $C = (T, \phi, \Pi, L)$ where:

- T represents the set of tuples (u, i, p) where $u \in U$ users, $i \in I$ items, and $p \in \mathbb{R}$ is a preference (rating) value.
- ϕ is a query on T.
- Π is $\{\varepsilon, \mathcal{R}, \alpha\}$ where ε is the extraction function on T, $\mathcal{R}$ is the recommender function, and α is the collaborative filter similarity algorithm in place.
- The set L contains the computed recommendation tuples (u, r, w) where $u \in U$ users, $r \in I$ items, and $w \in \mathbb{R}$ is a weighted preference value.

The collaborative filter extraction function $\varepsilon(T, \phi)$ produces the set G where ϕ is a query on T such that every $(u, i, p) \in G$ satisfies ($\vdash$) the query pattern in ϕ.

$$G = \{(u, i, p) | (u, i, p) \in T, \phi \vdash \{(u, i, p)\}\}$$

$\mathcal{R}$ is a single collaborative filter recommendation function that returns the set of recommendations for a single user u.

$$\mathcal{R}(u) = \{(u, r, w) | ((u, i, p) \in G \wedge (u, r, w) \notin G) \rightarrow (w = \alpha(i, r) \times p)\}$$

The collaborative filter similarity algorithm α returns a computed similarity value between an i and r (both from I) based on the preferences for them by users in G. The collaborative filter system produces recommendations for G as a candidate list L for each and every $(u, i, p) \in G$ based on the similarity (α) of each i it has a preference p for.

$$L = \bigcup_{(u,i,p) \in G} \mathcal{R}(u)$$

Content-based The content-based recommender determines the semantic-distance between item entities by referring to the features for similarity computation. This approach *vectorises* each entity and its features (e.g. RDF properties and values) to compute the semantic-distance between each item [7]. Content-based recommendation is $S = (T, \phi, \Pi, L)$ where:

- T represents the set of $i \in I$ items.
- ϕ is a query on T.
- Π is $\{\varepsilon, \mathcal{R}, \Delta, \Psi\}$ where ε is the extraction function on T, $\mathcal{R}$ is the recommender function, Δ is the semantic-distance algorithm in place, and Ψ represents the vectorisation of an item and its features.
- The set L contains the computed recommendation tuples (i, r, w) where $i \in I$ are items, $r \in I$ are the recommended items, and $w \in \mathbb{R}$ is a computed similarity value from Δ.

The content-based recommender extraction function $\varepsilon(T, \phi)$ produces the set G where ϕ is a query on T such that all $i \in G$ satisfies ($\vdash$) the pattern selection in ϕ.

$$G = \{i | i \in T, \phi \vdash i\},$$

$\mathcal{R}$ is a single content-based recommendation function that returns the set of recommendations for a single i.

$$\mathcal{R}(i) = \{(i, r, w) | (i \in G \wedge r \in G) \rightarrow (w = \Delta(\Psi(i), \Psi(r)))$$

The content-based system produces recommendations for G as a candidate list L for each and every $i \in G$ based on its similarity (Δ) to other items, $r \in G$.

$$L = \bigcup_{i \in G} \mathcal{R}(i)$$

2.1 System Framework and Workflow

In the adaptation of our formal definitions for the actual development of the recommender, the decision was for the system to execute in a *pipeline* fashion. Figure 1 shows the interconnectivity and flow of the pipelined modules.

The majority of the system was developed in the Java programming language. The backend system consists of a number of modules that generally perform preparation and processing of RDF data for the core recommender. Inside the core recommender, modules compute recommendations in parallel. The Apache Mahout [7] machine learning library is integrated into the recommender to perform the parallel and distributed algorithmic computations. This library contains a collection of the most prevalent recommendation algorithms for collaborative filters and content-based recommendations[2] and includes an API developed for use with Apache Hadoop[3].

[2] Mahout's Algorithms – `https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms`

[3] Apache Hadoop – `http://hadoop.apache.org`

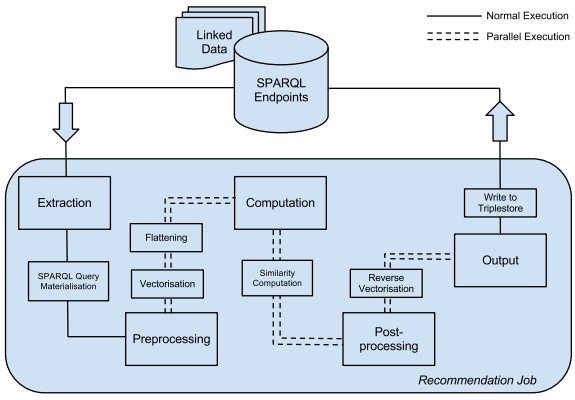

Figure 1. Recommendation Job Workflow

The front-end interface[4] of the system was developed in HTML and Javascript and utilises a REST API to interact with the backend system. This API consists of a Java servlet that directs communication from the user to the backend.

We describe the most prominent modules (or phases) involved with the processing of RDF data and the computation of recommendations.

Recommendation Jobs A singular recommendation *job* represents the input, computation, and output of recommendations for a set of data. In other words, the flow depicted in Figure 1. When a job is created, a recommender *config* is also created and acts as a backing towards the recommender job. A config contains all the settings for the job including the original SPARQL query and triplestore endpoint for the recommender input, the output triplestore endpoint to store the recommendations, options for the recommendation algorithm (based on Mahout's algorithms), plus other essential configuration details.

Extraction – $\varepsilon(T, \phi)$ When a recommender job is created and started, the first step of the workflow is to perform the extraction of data from an RDF triplestore. The recommender system utilises the OpenRDF Sesame Java API[5] for the execution of SPARQL queries and the retrieval of results. The API executes the defined query by synchronously communicating with a triplestore through the HTTP protocol. For either collaborative filters or content-based recommendations, this module will retrieve the query results and materializes them for preprocessing.

Preprocessing In this phase, data retrieved through the SPARQL query is transformed to efficiently organize the large volumes of RDF that will be input into the following recommendation computation modules.

In the case of collaborative filters, data is processed into a vector data structure. The first preprocessing step is to produce a dictionary for the results from extraction. These dictionaries are used to numerically map and structure the RDF into distinct vector representations.

[4] A demo of the system interface is available at `http://www.youtube.com/playlist?list=PL82ECC533A5936472`

[5] OpenRDF Sesame API – `http://www.openrdf.org/doc/sesame2/api/`

Algorithm 1 Preprocessing for Collaborative Filtering

```
procedure MAPPER(G)
    for all (u,i,p) in G do
        userkey = userdictionary.add(u)
        itemkey = itemdictionary.add(i)
        vectormap.push<userkey, itemkey, p>
    end for
    return vectormap
end procedure
```

With content-based recommendations, similar dictionary and mapping preprocessing jobs are performed to restructure the RDF data into sparse vectors. However, in addition, a *flattening* job ensures that non-unique entities returned from the SPARQL query are combined into singular results (Section 3 demonstrates the need for this).

Algorithm 2 Preprocessing for Content-based Recommendations

```
procedure MAPPER(G)
    for all i in G do
        itemkey = itemdictionary.add(i)
        vectormap.push(<itemkey, i.featuresList>)
    end for
    return vectormap
end procedure
```

Algorithm 3 Flattening for Content-based Recommendations

```
procedure REDUCER(vectormap)
    for all <k, featuresList> in vectormap do
        if sparsevector.get(k) then
            sparsevector.get(k).concatenate(featuresList)
        else
            sparsevector.put(k).concatenate(featuresList)
        end if
    end for
    return sparsevector
end procedure
```

Computation – $\mathcal{R}, \alpha, \Delta$ For collaborative filters, Mahout's *RecommendationJob* API [7] is initialised with inputs and configuration settings for the purpose of executing a job (either locally on a pseudo-cluster or on a real Hadoop cluster). For content-based recommendations, a similarity job is instead created using Mahout's *RowSimilarityJob* API [7], but is generally configured in the same way as the collaborative filter. Both API's are input with the preprocessed vectors from the previous phase and produce recommendations in a similar data structure format.

Post-processing and Output Due to the preprocessing of the input RDF data performed earlier, this post-processing step is required so that the referenced vector keys in the computed recommendation are transformed back into the original data. This ensures that data – such as URI's or literal values – can be output correctly as RDF recommendations. In general, the post-processing MapReduce job resembles the preprocessing algorithm in reverse.

With the recommendations computed and post-processed, the recommender system outputs these results as RDF on to the user-specified triplestore. Once again, the OpenRDF Sesame API is used for interacting with the triplestore. In this case, each result from the recommendation job is written as a new triple[6] into an RDF graph specific to the job. The

[6] The vocabulary for RDF recommendations is user-defined.

following is an example of a single recommendation for a collaborative filter; `User 1` has a recommendation for `movie 7` with a `rating` (score, similarity) of `72.441532`.

```
<http://user/1> :hasRecommendation :node16sus6lh.
:node16sus6lh :recommends <http://movie/7>.
:node16sus6lh :hasScore 72.441532.
:node16sus6lh rdf:type :Recommendation.
```

Each produced recommendation is added to the graph and committed to the triplestore using SPARQL and HTTP. Additional data is also written into the result graph. This includes the number of recommendations produced as well as the beginning and end time of the job.

3 System Use

To demonstrate the input and output of the recommender, we present a couple of use cases. The first example is for a collaborative filter. In this case, a SPARQL query is written to select all the *users* from a database who have indicated a preference to a *movie* with a *rating*[7].

```
SELECT ?user ?movie ?rating WHERE {
  ?user <http://www.grouplens.org/rating/hasRated> ?bn.
  ?bn <http://www.grouplens.org/rating/target> ?movie.
  ?bn <http://www.grouplens.org/rating/value> ?rating.
}
```

The resulting data, a triple of $(user, movie, rating)$, is used to determine probable ratings for the movies users have not specified ratings for yet. It is important to note the structure of the `SELECT` queries output; although the user is free to declaratively specify the SPARQL query in any way they wish (i.e. as long as it satisfies the RDF structure of the data graph), to properly function as a collaborative filter, the user must adhere to some restrictions when defining the output (i.e. projection) of the final query. The output must contain the following data in this order:

1. A subject, user, or entity that is portraying a preference towards something (i.e. another entity).
2. The item or other entity that the previous subject, user, or entity is showing a preference towards.
3. A weight or level of the preference towards the item [optional].

The third parameter is optional. In the case of its absence, the preference is presumed to be an atomic value (i.e. the user simply demonstrates a preference with no weight). For the above example, an excerpt of the queries output is shown in Table 1 (* = http://www.grouplens.org).

?user	?movie	?rating
*/user/1	*/movie/2115	3
*/user/1	*/movie/1240	5
*/user/2	*/movie/780	2
*/user/2	*/movie/1240	5

Table 1. Query Results for Collaborative Filter

Users can have any number of preference data (e.g. `user 1` has two preferences shown here). All the queries output data is compiled together as input for the system so that the recommender can determine the preference similarities between all users and can compute probable rating values for movies (i.e. those results returned by `?movies`) that the user has not yet shown a preference towards. For example, the recommender produces this RDF recommendation once computation is complete:

```
<*/user/1> :hasRecommendation :node79hjk1jk.
:node79hjk1jk :recommends <*/movie/780>.
:node79hjk1jk :hasScore 1.89712.
:node79hjk1jk rdf:type :Recommendation.
```

`User 1`'s probable preference to `movie 780` is quite low (in this case, preferences are in the 1-5 range). While still taking into consideration other input data, this result is possibly due to his similarity with `user 2` and his preference to `movie 780`. In regards to linked data, we can see that this result has retained links back to the original data by referencing the original URI's of the user and movie entities.

For a content-based recommendation, the following SPARQL query can be used to specify a number of items for similarity comparison.

```
SELECT ?person ?birthDate ?occupation ?abstract
FROM <http://dbpedia.org>
WHERE {
  ?person <http://dbpedia.org/ontology/occupation> ?occupation.
  ?person <http://dbpedia.org/property/dateOfBirth> ?birthDate.
  ?person <http://dbpedia.org/ontology/nationality> ?birthPlace.
  ?person <http://dbpedia.org/ontology/abstract> ?abstract.
  FILTER langMatches( lang(?abstract), "en" )
}
```

Here, entities of *people* recorded in DBpedia's RDF repository (http://dbpedia.org/sparql) represent the items of comparison. For qualifying features, we specify predicates and their values (URI or literal) as input. The data input, a tuple of $(person, birthDate, occupation, abstract)$, would be used for comparison and computation of similarity values which in turn could be used for the purpose of recommendation.

Unlike collaborative filters, queries used as input for content-based recommenders in our system do not have the same restrictions in regards to the actual output of the query. There is only one guideline when declaratively specifying the query; the first projection variable of the query must represent the distinct entities that you want to perform similarities between. In the case of the above example, we have written a query for the DBpedia.org SPARQL endpoint that retrieves any entities (`?person`) that have the following features: an occupation, a birth date, a birth place, and finally an abstract description about themselves. Table 2 shows an excerpt of this queries output (* = http://dbpedia.org/resource).

?subject	?birthDate	?occupation	?abstract
*/Nicanor_Parra	5	*/Poet	''Nicanor Parra Sandoval (born September 5, 1914) is...
*/Nicanor_Parra	5	*/Physicist	''Nicanor Parra Sandoval (born September 5, 1914) is...
*/Eduardo_Frei_Ruiz-Tagle	1942-06-24	*/Civil_engineering	''Eduardo Alfredo Juan Bernardo Frei Ruiz-Tagle (born June 24, 1942) is a...
*/Eve_Langley	1908-09-01	*/Poet	''Eve Langley (1 September 1908 – circa 1 June 1974), born Ethel Jane Langley, was...

Table 2. Query Results for Content-based Recommendation

After computation, one of the results for this recommenda-

[7] All the movie preference data was adapted from the `http://www.grouplens.org/` research datasets into RDF triples for the purpose of our experimentation.

tion job had the following RDF triples:

```
<*/Eve_Langley> :hasRecommendation :node45rlb8ui.
:node45rlb8ui :recommends <*/Nicanor_Parra>.
:node45rlb8ui :hasScore 72.12791.
:node45rlb8ui rdf:type :Recommendation.
```

This particular result was most likely due to these entities sharing the same occupation feature and also possibly having similarities in their abstracts (e.g. month of birth). Again, these results demonstrate how the original linked data is augmented through references to their URI's.

There are some important items to take notice of for content-based recommendations. First, note that not all results of SPARQL queries have to be used for input into the recommender. In the above example, although we queried subjects with a `?birthPlace` feature, we were not required to use it as input. This is useful if we require filters on the data we want to produce recommendations for but are not particularly concerned with its use. Furthermore, unlike queries used for collaborative filters, we are not limited to the number of entity features/properties used as input. This means we can broaden (or narrow) the similarity by selectively choosing the features we require for recommendation computation.

The output of the above query is an interesting example as it highlights a certain tendency of SPARQL query output; in the case of the first entity (the DBpedia resource for 'Nicanor Parra'), this particular resource satisfies the query two times. This is because 'Nicanor Parra' has a triple in the DBpedia data graph which states he has two occupations: Poet and Physicist. This is important as it demonstrates one of the preprocessing steps that the recommendation system had to implement (earlier we referred to it as a *flattening* procedure). The system aggregates these features and performs processing so that multiple query results like those of 'Nicanor Parra' are transformed into singular results.

It is also important to take note of the variety of unexpected output that can occur with the returned query results. In this example, the `?birthDate` feature is not consistent as we can see that the results vary in format. Another consideration comes from the fact that the `?abstract` feature contains a long string of literal data as opposed to a URI. Fortunately, options are built into the recommenders preprocessing modules to handle these considerations (e.g. date and string tokenization). Due to space constraints, we will forgo discussing them in full detail.

4 Hybrid-based Recommendations

Hybrid-based recommenders are a combination of different recommender types for the purpose of producing more precise results [3]. The ability to create hybrid recommenders are an important feature of recommendation systems. They can help alleviate recommendation computation when data is very sparse (i.e. the *cold-start* problem) [3]. In some cases, hybrids can be utilised as "tie-breakers" when one recommender does not produce relevant enough results. The limitations of hybrid recommenders are based on the ability to interconnect the functionality or input/output of individual recommenders.

Our hybrids approach is based on the reuse of linked data recommendations created from previous jobs. In [3], Burke gives a number of strategies and descriptions of hybrid recommenders which in some cases are used as a basis for our own. However, our current hybrids are not exclusively limited to his hybrid types as we were able to define our own variations. The first hybrid we defined creates a union between the most relevant recommendations computed by a collaborative filter for a user u and the most relevant recommendations computed by a content-based recommendation for a set of items i.

Definition Simple combination:
Assuming there exists a $G = \varepsilon(T, \phi)$ and L for collaborative filtering, and a $G' = \varepsilon(T, \phi')$ and L' for content-based, for a u and i, a simple combination of their recommendations is $H = \{r|(u, r, w) \in L\} \cup \{r'|(i, r, w) \in L'\}$, where a limiting bound can be set on w (e.g. $l \leq w \leq m$ where l is a lower bound and m is an upper bound).

We also present two types of *cascade* hybrids [3]. These hybrids involve the direct refinement of the results of one recommender within another.

Definition Cascade refinement (CF, CB):
Assuming there exists a $G = \varepsilon(T, \phi)$ and L for collaborative filtering and a $G' = \varepsilon(T, \phi')$ and $L'$for content-based, for a u, retrieve the collaborative filter recommendations: $H = \{r|(u, r, w) \in L\}$. A refinement of the recommendation H is $H' = \{(r, w'')|r \in H \wedge w'' = \sum w' \in T(r, H)\}$, where $T(r, H) = \{w'|\forall i \in H((i, r, w') \in L' \wedge i \neq r)\}$. For each r in H', a total weight (score) w'' is associated with it. A limiting bound can be set on w'' (e.g. $l \leq w'' \leq m$ where l is a lower bound and m is an upper bound).

Definition Cascade refinement (CB, CF):
Assuming there exists a $G = \varepsilon(T, \phi)$ and L for content-based, for a u, return the set of most similar users $F = \{u'|(u, u', w) \in L\}$. For the refined recommendation, a query ϕ' can be made for the set $(u \cup F)$ and all of their preferred $i \in I$ where $I \subset T$; that is $G' = \varepsilon(T, \phi')$. L' is the refined collaborative filter for u based on its semantic distance to F.

These hybrids refine the results of a collaborative filter through content-based recommendation and vice-versa. The first cascade hybrid refines the results of a collaborative filter by utilising the content-based recommendations of the entities to compute the sum of the similarity weights between them. Each sum represents a new ranking of the entities and is tupled together.

The second cascade hybrid in contrast computes the semantic-distance of users with a content-based recommender (possibly using feature combination of user attributes) so that a collaborative filter can be executed for a particular user, their *neighbourhood* of similar users (F), and their preferred items. A SPARQL query (ϕ') is written to reflect this and the output is a refinement for a broader collaborative filter over, for example, all users of a triplestore T.

5 Related Work

Implementations of recommendation engines specific to linked data is something that has been explored before. However, we believe our approach is quite different than most primarily due to the integration of SPARQL as a method for the declarative

specification of recommender input, its ability to create new linked data, and the use of a distributed programming model for recommendation computation.

In [8], Passant presents his theory and implementation for *dbrec*, a music recommendation system using data from DBpedia. His algorithm, which is primarily based on SPARQL query patterns, disregards literal values contained within the data entities used to compute recommendations. We consider all values (features) of semantic data necessary to compute similarities and eventual recommendations. However, many of the motivations of this work are also aligned with ours, including using linked data as a way to enhance both the recommendation input and output.

In [5], Heitmann and Hayes discuss their work on a collaborative filter recommendation system that utilises linked data in order to improve the recommendations. They propose a system which integrates linked data with *closed recommendation data* for the purpose of the data acquisition problem (i.e. not having enough data to compute relevant recommendations). Like our approach, they utilise SPARQL for the purpose of integrating linked data. However, our implementation not only uses linked data as input but also produces recommendations as new linked data. In addition, our system can be utilised to create content-based as well as hybrid recommendations.

A comprehensive exploration is done in [4]. Cantador et al. implement an Internet news recommendation system based on their research with semantic ontology based knowledge models and recommenders. Their approach also covers hybrids between collaborative filters and content-based recommenders. A similar framework was designed (e.g. RDF data storage, machine learning library use), but is not based on modern technologies such as RDF triplestores or SPARQL. Furthermore, although the works focus is on semantic data in general, the implementation was based solely on RDF Site Summary (RSS) news feeds.

Other work has shown some innovation within the more general area of machine learning and linked data. For example, in [11], Stankovic et al. demonstrate the use of recommendation techniques and the linked open data cloud, specifically DBpedia, for computing similarities and allocating topics or keywords to higher-level concepts.

6 Future Work and Conclusion

In this paper we have demonstrated a practical implementation of a SPARQL integrated recommender for linked data. Due to the capability to not only compute recommendations with linked data but to also create and augment linked data, we deemed the exploration and development of such an engine worthwhile. This idea garnered further potential with the decision to utilise the SPARQL query language as a user-oriented way for declaratively specifying recommender input. In regards to the ubiquitous and generally large volume of linked data that can be expected to be used with a recommendation system, our implementation was designed for parallel recommendation computation and scalability through the use of the Mahout machine learning library.

After presenting formal definitions of our recommendation engine, we elaborated on the design and development of our systems framework. Case studies were given to demonstrate the use of SPARQL and the RDF output of our recommender. Based on the idea of hybrid-based recommenders, we formally defined and showed how our system can interconnect the output of different recommendations to produce refined computations. Finally, we presented a brief review of the relevant state of linked data recommendation systems.

With future development we plan on continuing to improve our framework to accommodate different use cases. Communication and collaboration with small to medium sized enterprises (SME) has helped to establish these and future use cases and to define desired system features. In addition, we will explore the integration of other recommender types into the system. Experimental benchmarking is a task that will be undertaken before any future work is accomplished. However, this has proved to be quite difficult as access to linked data resources can sometimes be erratic or unreliable. Currently, most tests have been completed with either large volumes of local data or small volumes from externally served RDF triplestores or search engines (e.g. DBpedia.org, Sindice.com).

7 Acknowledgements

This research is funded by the Semantic Tools for Digital Libraries Project[8] (SemLib) and the LOD2 Project[9].

REFERENCES

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[2] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1265–1266, New York, NY, USA, 2008. ACM.

[3] Robin Burke. Hybrid web recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[4] Iván Cantador, Pablo Castells, and Alejandro Bellogín. An Enhanced Semantic Layer for Hybrid Recommender Systems. *International Journal on Semantic Web and Information Systems*, 7(1):44–78, MarJan 2011.

[5] Benjamin Heitmann and Conor Hayes. Using linked data to build open, collaborative recommender systems. In *In: AAAI Spring Symposium: Linked Data Meets Artificial Intelligence'. (2010*, 2010.

[6] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of Machine Learning*, pages 829–838. 2010.

[7] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications, 2011.

[8] Alexandre Passant. dbrec - music recommendations using dbpedia. In *International Semantic Web Conference (2)*, pages 209–224, 2010.

[9] A. Rajaraman and J.D. Ullman. *Mining of massive datasets*. Cambridge Univ Pr, 2011.

[10] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[11] Milan Stankovic, Werner Breitfuss, and Philippe Laublet. Linked-data based suggestion of relevant topics. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 49–55, New York, NY, USA, 2011. ACM.

[8] `http://www.semlibproject.eu/`

[9] `http://lod2.eu/`

Bridging the Gap between RIF and SPARQL: Implementation of a RIF Dialect with a SPARQL Rule Engine

Oumy Seye[1] and **Catherine Faron-Zucker**[2] and **Olivier Corby**[3] and **Corentin Follenfant**[4]

Abstract. Semantic Web knowledge representation standards have close relationships with rule-based languages and systems. In particular, the SPARQL query language can be seen as a rule language: answering a CONSTRUCT query is similar to applying a rule in forward chaining to enrich an RDF base, with the rule antecedent corresponding to the WHERE clause and its consequent to the CONSTRUCT clause of the SPARQL query. In this paper we present the correspondances between SPARQL and RIF, the rule interchange format recommended by the W3C. We have characterized the subset of SPARQL that can be expressed in RIF and, conversely, we have searched for the maximal RIF dialect that can be expressed in SPARQL. We have extended the Corese semantic engine which enables to query RDF/S data with SPARQL and is provided with both a forward and a backward chaining rule system, so that it now supports RIF: we have developed (1) a RIF parser which builds an internal representation of a RIF rule into an abstract syntax tree (AST) and (2) a translator of a RIF AST into a SPARQL AST so that Corese has become an implementation of a RIF-SPARQL dialect.

1 Introduction

Semantic Web knowledge representation standards have close relationships with rule-based languages and systems. The semantics of RDFS and that of some subsets of OWL - in particular OWL2 RL - can be axiomatized in the form of first order logic implications that can be used as the basis for a rule-based implementation. Also, the coupling of a rule base and a light vocabulary (in RDFS) appears as a real alternative to OWL vocabularies and Description Logic reasoners whose complexity of operations is quite high. Finally, the SPARQL query language for RDF restricted to the CONSTRUCT form can be seen as a rule language: answering a CONSTRUCT query is similar to applying a rule in forward chaining to enrich an RDF base, with the rule antecedent corresponding to the WHERE clause and its consequent to the CONSTRUCT clause of the SPARQL query.

In this paper, we focus on the relationships between SPARQL, the query language for RDF recommended by the W3C[5], and RIF[6], the rule interchange format recommended by the W3C for the exchange of any kind of rules on the web and more specially on the semantic web. RIF has been designed to ensure the interoperability and portability of different rule languages and systems; it enables the exchange and reuse of rules by different rule engines. RIF is an extensible set of dialects, three of which are defined in the recommendation: RIF-BLD[7], RIF-PRD[8] and RIF-CORE[9]. BLD stands for Basic Logic Dialect: RIF-BLD enables to represent logic programs, i.e. inferences rules on positive facts. It corresponds to Horn Logic with equality. Syntactically, it is extended with frames, URI denoting concepts and XML Schema data types. PRD stands for Production Rules Dialect: RIF-PRD enables to represent production rules. RIF-CORE is the core language made of the primitives common to RIF-BLD and RIF-PRD. It corresponds to the Horn Logic without function symbol, i.e to Datalog, with classical first-order logic semantics.

In the context of the semantic web, the question of the compatibility of RIF and RDF/S or OWL arises. The integration of RIF in the Semantic Web Stack and its actual use on the Web of Data requires the writing of RIF-BLD rules to be applied on RDF data and using RDFS or OWL vocabularies. A typical scenario is the exchange of RIF rules with RDF data and RDFS or OWL vocabularies between rule-based systems capable to take into account the semantics of the vocabularies while applying RIF rules on RDF data. Another scenario is that of a search engine taking into account the semantics of RIF-BLD rules while evaluating SPARQL queries on RDF data: it will answer not only facts present in the RDF dataset but also facts inferred from the dataset by the application of RIF rules. The SPARQL engines which implement the semantics of RDFS or OWL run inferences, for example to find every resources of a given type (finding also the resources declared with subtypes). In particular, the Corese engine which we develop in our team enables to query RDF/S data in the SPARQL language and implements the semantics of RDFS and some OWL primitives. It is provided with both a forward and backward rule-based system to perform inferences over an RDF/S dataset to answer queries. Rules are viewed as part of the ontology, complementing RDFS or OWL statements. They are represented in

[1] INRIA Sophia-Antipolis Méditerranée, France
[2] Laboratoire I3S, Université Nice Sophia Antipolis, France
[3] INRIA Sophia-Antipolis Méditerranée, France
[4] INRIA Sophia-Antipolis Méditerranée, France

[5] http://www.w3.org/TR/rdf-SPARQL-query/
[6] http://www.w3.org/TR/2010/NOTE-rif-overview-20100622/
[7] http://www.w3.org/TR/rif-bld/
[8] http://www.w3.org/TR/rif-prd/
[9] http://www.w3.org/TR/rif-core/

the SPARQL language, using the CONSTRUCT query form.

After the publication of the RIF recommendation, we were interested to bring the rule engine on RIF-BLD. Then we tried to define the maximal subset of RIF-BLD which can be expressed in SPARQL. We have implemented both a RIF parser which builds an internal representation of a RIF rule in an abstract syntax tree (AST) and a translator of a subset of this AST into the SPARQL AST handled by Corese. As a result, Corese provides an implementation of a RIF-SPARQL dialect.

The paper is organized as follows. In Section 2 we present preliminary material. Section 3 is dedicated to the translation of SPARQL into RIF and the translation of RIF-BLD into SPARQL and the identification of the corresponding RIF-SPARQL dialect. Section 4 describes our implementation work within the Corese engine and the results we obtain on the W3C test database.

2 Preliminaries

2.1 RIF-BLD

2.1.1 The RIF-BLD language

RIF-BLD is provided with two syntaxes: an abstract syntax which will be used throughout this paper and a concrete XML syntax which is the interchange format. In the abstract syntax, as it is conventional in mathematical logic, the W3C recommendation defines RIF-BLD as a set of well formed formulas built with terms based on an alphabet.

The alphabet of RIF-BLD comprises: a set of constant symbols $Const$; a set of variable symbols Var disjoint with $Const$, all variable symbols starting with '?'; a set of argument names $ArgNames$ disjoint with $Const$ and Var; connective symbols And, Or and $:-$; existential and universal quantifiers $Exists$ and $ForAll$; a few other symbols that will be introduced in the following as and when they are used.

A RIF-BLD term is either:

- a simple term, i.e. a constant or a variable,
- a positional term $t(t_1, \dots t_n)$ where t is a constant and $t_1, \dots t_n$ are terms,
- a term with named arguments $t(s_1 -> v_1, \dots s_n -> v_n)$ where t is a constant, $v_1, \dots v_n$ are terms and $s_1, \dots s_n$ are argument names,
- a list of terms either closed: $List(t_1 \dots t_m)$, or open: $List(t_1 \dots t_m|t)$,
- an equality between base terms $t_1 = t_2$,
- a class membership $t_1\#t_2$, where t_1 represents an object and t_2 a class,
- a class specialization $t_1\#\#t_2$, where t_1 and t_2 represent classes,
- a frame $t[p_1 -> v_1, \dots p_n -> v_n]$, where t, $p_1, \dots p_n$, $v_1, \dots v_n$ are terms,
- an external term External(t) where t is a positional term or a term with named arguments.

A positional term corresponds to an atomic formula in first order logic. In a term with named arguments, symbol t represents a predicate or a function. A frame represents an object.

ARIF-BLD atomic formula is either a positional term, a term with named arguments $t(s_1 -> v_1, \dots s_n -> v_n)$ where t represents a predicate, an equality, a class membership, a class specialization, a frame or an external terms External(φ) where φ is an atomic formula. Others terms — constants, variables and lists — are used for building formulas but they are not formulas.

The set of RIF-BLD formulas includes:

- atomic formulas,
- conjunctive formulas $And(\varphi_1, \dots \varphi_n)$ where $\varphi_1, \dots \varphi_n$ are atomic or conjunctive or disjunctive or existential formulas,
- disjunctive formulas $Or(\varphi_1, \dots \varphi_n)$ where $\varphi_1, \dots \varphi_n$ are atomic or conjunctive or disjunctive or existential formulas,
- existential formulas $Exist\,?v_1, \dots ?v_n\,(\varphi)$, where $?v_1, \dots ?v_n$ are variables and φ is an atomic or conjunctive or disjunctive or existential formula,
- rule implications $\varphi :- \psi$, where φ is an atomic formula or a conjunction of atomic formulas not externally defined and where ψ is an atomic or conjunctive or disjunctive or existential formula; φ is the conclusion of the rule and ψ is its premise,
- universal rules $Forall\,?v_1, \dots ?v_n\,(\varphi)$, where $?v_1, \dots ?v_n$ are variables and where φ is an implication rule whose free variables are among $?v_1, \dots ?v_n$; universal rules are also called RIF-BLD rules,
- universal facts $Forall\,?v_1, \dots ?v_n\,(\varphi)$, where $?v_1, \dots ?v_n$ are variables and where φ is an atomic formula whose free variables are among $?v_1, \dots ?v_n$; universal facts can be seen as universal rules without premise,
- groups of universal facts, variable-free rule implications, variable-free atomic formulas, or group formulas,
- documents embedding an optional group formula and an optional sequence of directives which are not detailed in this paper.

A RIF-BLD formula is well formed if and only if each of the constant symbols it uses belongs to a different context, i.e. represents either an individual or a function or a predicate or an external predicate or an external function.

2.1.2 Reasoning in RIF-BLD

Reasoning in RIF-BLD consists of deriving new facts by applying inference rules: universal instantiation (instantiation of universal rules) and modus ponens, and by evaluating conjunctive or disjunctive formulas.

Instantiation. Consider for example the universal rule representing the well known statement that "every man is mortal":

```
Forall ?x (?x # bio:Mortal :- ?x # bio:Human)
```

This universal rule can be instanciated by substituting for its variable `?x` any IRI representing an individual. For example, by substituting `phil:Socrates` representing a famous man for `?x`, the following rule is obtained:

```
phil:Socrates # bio:Mortal :- phil:Socrates # bio:Human
```

Modus ponens. Let us now consider a document embedding the above universal rule and a positional term representing the fact that "Socrates is a man":

```
Group( Forall ?x (?x # bio:Mortal :- ?x # bio:Human)
       phil:Socrates # bio:Human )
```

Once the universal rule has been instanciated by replacing its variable `?x` by the constant `phil:Socrates`, its premise can be asserted to be true, regarding the positional term in the document. The modus ponens inference rule then enables to deduce a new formula, `phil:Socrates#bio:Mortal`, corresponding to the conclusion of the instanciated rule and representing the fact that "Socrates is mortal".

Evaluation of conjunctive and disjunctive formulas. For non-atomic rule premises, evaluation rules are applied to evaluate their truth value: a conjunctive formula is true if every formula in the conjunction is true; a disjunctive formula is true if at least one of the formulas in the disjunction is true.

Operationalization. These three pieces of reasoning described above enable to operationalize the semantics of RIF-BLD either with the reasoning method by *forward chaining* which consists of recursively handling rules, starting with their premises to conclude new facts which enrich the knowledge base, or with the reasoning method by *backward chaining* which consists of proving a fact by recursively handling rule conclusions to come back to the facts in the knowledge base.

2.2 The SPARQL Rule Language

SPARQL is the query language for RDF. It is designed to meet the requirements of the directed labeled graph model of RDF: a SPARQL query basically contains a graph pattern which is a set of triples, like RDF triples, whose subject or predicate or object may be variables. A SPARQL query is evaluated on an RDF graph by matching its graph pattern with subgraphs of the RDF graph, RDF terms of this subgraph being substituted for the variables of the SPARQL query graph.

SPARQL has several query forms among which the SELECT query form is the most common, that returns variable bindings. More precisely, it returns the bindings of the variables in the SELECT clause of the query which enable to match the query graph pattern in the WHERE clause of the query with subgraphs of the RDF graph. In addition, the ASK query form enables to test wether a query graph pattern has a solution, i.e. matches any subgraph of the RDF graph. ASK queries can then be used to represent constraints. Finally, the CONSTRUCT query form returns an RDF graph specified by a graph template in the CONSTRUCT clause and instanciated with the variable bindings which enable to match the graph pattern in the WHERE clause of the query with the RDF graph.

Such a CONSTRUCT query can be viewed as a rule and its processing as the application of the forward chaining inference rule to enrich the RDF graph. Here is the representation in SPARQL of the rule "Every man is mortal":

```
CONSTRUCT {?x rdf:type bio:Mortal}
WHERE {?x rdf:type bio:Human}
```

SPARQL 1.0 predates RIF and has been used as a rule language in several works on the Semantic Web. This enables to directly handle RDF triples in the premises and conclusions of rules applied on RDF datasets. The Corese engine [2] embeds two rule engines, one in forward chaining and another one in backward chaining, both of them sharing a parser for rules represented in SPARQL.

SPIN[10] stands for SPARQL Inferencing Notation. It proposes an RDF model to represent SPARQL rules and constraints (CONSTRUCT, UPDATE and ASK query forms). With SPIN, rules are therefore expressed in SPARQL. This RDF notation of SPARQL aims at integrating rules and constraints in a knowledge base with a unified knowledge representation language. SPIN is a W3C member submission since 2011[11].

A. Polleres provides a translation from SPARQL to Datalog, the query and rule language for deductive databases [5]. He concludes that SPARQL can serve as an expressive rule language on top of RDF. Angles and Gutierrez further study the expressive power of SPARQL compared with Datalog [1].

With SPARQL++, A. Polleres et al. use SPARQL as a rule language to express alignments between RDF vocabularies and they propose some extensions to the CONSTRUCT query form for their specific purpose [6].

3 From SPARQL to RIF and back

In this section we first present a translation of the subset of SPARQL that can be used as a rule language into RIF, now the standard rule interchange format. Conversely, we propose a translation of RIF into SPARQL. Our motivation in this second translation is that there is still very few implementations of RIF (let us mention [4] which bases on Datalog), whereas there is a wide range of implementations of SPARQL, among which a number of them handle SPARQL rules.

3.1 From SPARQL to RIF

SPARQL queries in the CONSTRUCT form can be used to represent rules and therefore are candidates to be translated into RIF for interchange on the web. More precisely, if the WHERE clause of a SPARQL query contains any pattern other than a basic graph pattern or a union of basic graph patterns, then the query cannot be translated into RIF. Else it is translated into a universal RIF rule as follows:

- the conclusion of the RIF rule is the conjunction of all the RIF terms translating the triples in the graph template of the CONSTRUCT clause of the SPARQL query;
- if the WHERE clause of the SPARQL query contains a basic graph pattern(BGP) which is the conjunction of triples with possibly filters, then the premise of the RIF rule is the conjunction of all the terms translating the triples and the filters in the graph pattern; if the WHERE clause of the SPARQL query contains a union of basic graph patterns, then the premise of the RIF rule is the disjunction of the translations of these basic graph pattern;
- the variables in the quantification part of the RIF rule are those of the graph pattern of the SPARQL query. A special case is that of a SPARQL rule without variable which is then translated into a RIF rule implication.

Let us now detail which RIF terms translate which triples in a graph template or a basic graph pattern. Our translation relies on the "RIF-BLD and RDF compatibility" defined in the RIF recommendation[12]: a triple in a SPARQL graph template

[10] http://spinrdf.org/
[11] http://www.w3.org/Submission/spin-SPARQL/
[12] http://www.w3.org/TR/rif-rdf-owl/

or pattern is nothing else than an RDF triple with possibly variables. The translation is as follows:

- a triple (o rdf:type c) is translated into a class membership o#c;
- a triple (c_1 rdfs:subClassOf c_2) is translated into a class specialization c_1##c_2;
- a set of triples representing an RDF list is translated into an equality to a closed list;
- any other triple (o p v), with p ≠ rdfs:subClassOf, p ≠ rdf:type, p≠rdf:first, p≠rdf:rest, is translated into a frame o[p−> v]; a set of triples having the same subject can be translated into a single frame o[p_1 −> v_1 ... p_n −> v_n].

In these triples, URI are translated into `rif:iri` constants, literals without datatype into constants of type `xs:string`, typed literals into corresponding constants, variables and blank nodes into variables. Let us note that triples p_1 rdfs:subPropertyOf p_2 have no special translating term in RIF.

The translation of SPARQL filters occuring in a graph pattern is as follows:

- a conjunction or disjunction of filters is tranlated into a conjunctive or disjunctive formula, i.e. the conjunction or disjunction of the translations of the filters;
- an equality test is translated into an equality term;
- other SPARQL operators and functions are translated into RIF external terms;
- the translation of filters preserves XSD datatypes.

Table 1 shows three examples of SPARQL queries and their translation into RIF rules.

Table 1. Translation of three SPARQL queries into RIF-BLD rules

SPARQL	RIF
CONSTRUCT {?x rdf:type ?z} WHERE {?y rdfs:subClassOf ?z ?x rdf:type ?y}	Forall ?x ?y ?z (?x#?z :- AND(?y##?z ?x#?y))
CONSTRUCT {?x ex:uncleOf ?z} WHERE {?x ex:brotherOf ?y ?y ex:parentOf ?z}	Forall ?x ?y ?z (?x[ex:uncleOf → ?z] :- AND(?x[ex:brotherOf → ?y] ?y[ex:parentOf → ?z]))
CONSTRUCT {?x rdf:type ex:Adult} WHERE {?x rdf:type ex:Person ?x ex:age ?y FILTER (?y >= 18)}	Forall ?x ?y ?z (?x#ex:Adult :- AND(?x#ex:Person ?x[ex:age → ?y] External(pred:numeric- greater-than-or-equal(?y 18))))

In addition, a base of SPARQL queries may come with an RDF base or, when considering SPARQL 1.1 Update, a base of INSERT DATA operations which both can be translated into conjunctive formulas: the conjunction of all the frames o[p−> v] translating the triples (o p v) in the RDF base or in the graph template which is the operand of INSERT DATA.

3.2 From RIF-BLD to SPARQL

Let us now consider the reverse translation of RIF-BLD rules into SPARQL, which will enable us to reason with standard rules within a SPARQL engine.

3.2.1 Normalization of RIF-BLD Formulas

To characterize the RIF dialect which can be translated into SPARQL, a preliminary step has consisted in normalizing RIF-BLD formulas to reduce their complexity. We normalized RIF-BLD formulas by applying classical transformations similar to those in [3] for the translation of a WSML ontology into Datalog rules or those in [4] for the translation of RIF-BLD formulas into Datalog rules. One of the most important transformations is the conversion of a RIF formula into its disjunctive normal form. Such a normalization enables to simplify the translation of RIF-BLD formulas into SPARQL: the translation of a RIF-BLD formula is then the composition of elementary translations applied recursively.

3.2.2 The RIF-SPARQL Dialect

Among RIF-BLD formulas, rule implications and universal rules are translated into SPARQL queries of the CONSTRUCT form. The premise of a RIF rule is translated into a graph pattern and its conclusion into a graph template (which is a special case of basic graph pattern with no filter). In other words, the atomic, conjunctive, disjunctive or existential formulas occuring in the premise or conclusion of RIF rules are translated into SPARQL graph patterns:

- a conjunctive formula is translated into a basic graph pattern
- a disjunctive formula is translated into a union of basic graph patterns;
- an atomic formula is translated as described in the following;
- existential quantification is automatic: the variables which do not appear in the universal quantification part of a rule are automatically interpreted as existentially quantified in the graph patterns tranlating its conclusion and its premise.

The translation function from RIF-BLD frames, class memberships, and class specializations to SPARQL basic graph patterns is the inverse of the translation function of basic graph patterns described in the previous section:

- membership, subclass, and frame terms are each translated into a triple;
- equality tests between a constant or variable and a list are each translated into a set of triples describing a list;
- equality terms involving two variables or constants or two lists are each translated into an equality test in a filter; other equality terms are not translated.
- External terms are translated into operators and functions in filters.

In addition, among the terms with named arguments and positional terms, predicates are translated into basic graph patterns as described in table 2. For this purpose, we defined an RDF vocabulary identified with the rs namespace prefix.

To sum up, when compared to RIF-BLD, the main restrictions in RIF-SPARQL is the exclusion of universal facts, of open lists and the limitation of terms to constants, variables and closed lists in Equal, Member, Subclass and Frame formulas .

Here is the EBNF grammar of RIF-SPARQL for its presentation syntax:

Table 2. Translation of RIF terms with named arguments and positional terms into SPARQL BGP

RIF-BLD terms	SPARQL BGP
$P(n_1 -> v_1 \ldots n_n -> v_n)$	_:b_n rdf:type rs:NamedArgs. _:b_n rs:name P. _:b_n rs:arity n. _:b_n n_1 v_1. _:b_n ... _:b_n n_n v_n
$P(t_1 \ldots t_n)$	_:b_n rdf:type rs:Positional. _:b_n rs:name P. _:b_n rs:arity n. _:b_n rs:arg_1 t_1. ... _:b_n rs:arg_n t_n

```
RULE        ::= CLAUSE |
                'Forall' Var+ '(' CLAUSE ')'
CLAUSE      ::= ATOMIC ':-' FORMULA |
                'And' '(' ATOMIC* ')' ':-' FORMULA
FORMULA     ::= ATOMIC |
                ('And' | 'Or') '(' FORMULA* ')' |
                'Exists' Var+ '(' FORMULA ')'
ATOMIC      ::=  Atom | Frame | Member | Subclass |
                 Equal
Atom        ::= UNITERM
UNITERM     ::= Const '(' TERM* ')' |
                Const '(' (Name '->' TERM)*) ')'
Equal       ::= TERM1 '=' TERM1
Member      ::= TERM1 '#' TERM1
Subclass    ::= TERM1 '##' TERM1
Frame       ::= TERM1 '[' (TERM1 '->' TERM1)* ']'
TERM        ::= TERM1 | TERM2
TERM1       ::= Const | Var | List
TERM2       ::= Expr | 'External' '(' Expr ')'
Expr        ::= UNITERM
List        ::= 'List' '(' TERM1* ')'
```

Facts, i.e. atomic formulas (not involved in the premise or conclusion of a rule) can be translated into insert data operations in SPARQL 1.1 Update, except for equality terms and predicates with arity greater than 2.

4 Implementation of RIF-SPARQL

In this section we present our implementation of RIF-SPARQL with the Corese semantic engine. Figure 1 presents its general architecture (in blue) and its integration with the SPARQL rule engine of Corese (in green). Starting with Corese provided with a SPARQL parser and a SPARQL rule engine, we developed a RIF parser producing a RIF AST and we implemented a translation function of the subtree of the RIF AST corersponding to the RIF-SPARQL dialect into the SPARQL AST of Corese.

4.1 RIF-BLD Parser

Since the recommandation provides a normative mapping from the RIF-BLD presentation syntax to XML, we designed an abstract syntax tree (AST) model to be shared by the presentation syntax and the XML syntax of RIF-BLD[13] and we designed a parser for each of its syntax.

[13] http://www.w3.org/TR/rif-bld/#sec-xml-bld

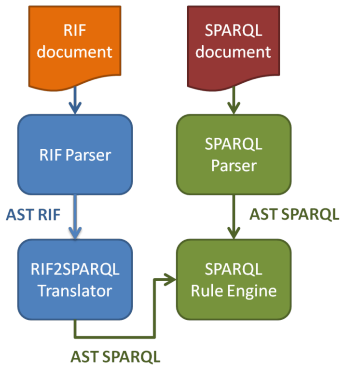

Figure 1. Architecture of our RIF-SPARQL implementation

The Java API of our parser is very simple. An abstract class `RIFDocument` is provided with three main methods: CREATE(), COMPILE() and GETPAYLOAD() which enable to create a `RIFDocument` from a file, compile it and get its AST. It is specialized by two concrete classes RIFPSDOCUMENT and RIFXMLDOCUMENT whose instances are created from files either in the presentation syntax or the XML syntax.

We used the JavaCC parser generator to generate a Java parser from the EBNF grammar of RIF-BLD presentation syntax[14]. Each grammar rule expressed in JavaCC is compiled into a Java method implementing the semantic actions described in the rule. Since the presentation syntax of RIF is a human-oriented syntax, its grammar cannot be efficiently interpreted (it does not belong to LALR(1) class). Our parser computes lookahead sets for some rules in the grammar.

We used the JAXB API to automatically build a Java class hierarchy from the normative XML schema for RIF-BLD[15]. To each class of this automatically built AST model, we added a method to connect it with the Java classes of the AST model manually designed. These methods enable to create instances of classes defined in the RIF AST model from instances of classes in the automatically built AST model and therefore to translate an AST in the automatically built model into an AST in the model manually defined. The JAXB API is used with this class hierarchy to parse a RIF document in the XML syntax and automatically generate its corresponding AST.

4.2 Corese SPARQL Rule Engine

Corese includes a backward and a forward rule engine and an implementation of SPARQL 1.1. We have implemented a partial conversion of the abstract syntax tree of a RIF document into the abstract syntax tree of SPARQL defined in Corese. The subtree of the AST model which can be translated corresponds to the RIF-SPARQL dialect we have defined.

RIF-BLD axioms are translated into SPARQL Update INSERT DATA operations that are run to construct the RDF graph on which RIF universal rules will be applied after their translation into SPARQL CONSTRUCT queries.

[14] http://www.w3.org/TR/rif-bld
[15] http://www.w3.org/TR/rif-bld/#sec-xsd-bld

Let us consider for example the RIF document taken from the positive entailment test of the W3C test base[16]. It comprises two axioms and one universal rule:

```
Group (
  Forall ?C ?I ?P ?V (
    ?I[?P->?V]
    :- And( ?C[?P -> ?V] , ?I # ?C ))
  ppl:john # cpt:Person
  cpt:Person[tax:phylum -> tax:Chordata] )
```

They are translated into the following two INSERT DATA operations and CONSTRUCT query:

```
INSERT DATA {ppl:john rdf:type cpt:Person}       (T1)
INSERT DATA {cpt:Person tax:phylum tax:Chordata}(T2)
CONSTRUCT { ?I ?P ?V }                          (Q1)
WHERE { ?C ?P ?V . ?I rdf:type ?C }
```

The two INSERT DATA operations are first performed to enrich an RDF graph or to construct a new RDF graph in case there is no additional RDF data. In forward chaining, the application of the CONSTRUCT query calls for the execution of the following ASK query (whose graph pattern is the graph pattern of the CONSTRUCT query, i.e. the premise of the rule):

```
SELECT * {?C ?P ?V . ?I rdf:type ?C}            (Q2)
```

When the graph pattern in the WHERE clause of this query is matched with the RDF graph, the query is answered the set of bindings {(?C, cpt:Person), (?P, tax:phylum), (?V, tax:Chordata), (?I, ppl:john)}. The triple in the CONSTRUCT clause of the query is then instanciated by substituting for its variables their binding values. This activates the materialization of the following triple in the RDF graph:

```
ppl:john tax:phylum tax:Chordata                (T3)
```

Triple `T3` is the instantiation of the graph template in the CONSTRUCT clause of query `Q1` by substituting for its variables their binding values found when matching the graph pattern in the WHERE clause of query `Q1`.

In backward chaining, let us for instance consider the evaluation of the following ASK query `Q3`:

```
ASK { ppl:john tax:phylum tax:Chordata }        (Q3)
```

It calls for the search of the CONSTRUCT queries which graph template in the CONSTRUCT clause matches with its graph pattern. For each query found, this in turn calls for the evaluation of an ASK query which graph pattern is built by instanciating the graph pattern of the query with the bindings found for the variables of the graph template of this query when matching it with query `Q3`. In our running example, the graph template of query Q1 matches with query Q3 and therefore the following ASK query is built and evaluated:

```
ASK { ppl:john rdf:type ?x .            (Q4)
      ?x tax:phylum tax:Chordata }
```

This query is answered `true` and then query `Q3` is answered `true` too.

4.3 Evaluation

To evaluate our method, we have used the RIF-BLD test cases proposed by the W3C RIF working group for positive entailment and negative entailment[17]. There are 26 positive entailment test cases and 4 negative entailment test cases for RIF-BLD. A positive entailment test case comprises a given RIF condition (the goal to prove) which is entailed by a given RIF document (rules and facts) and a negative entailment test case comprises a given RIF condition which is not entailed by a given RIF document.

Positive entailment test cases enable to test if a rule engine can find that a given goal is entailed by a given set of facts and rules. Among the 26 such cases in the base, Corese succeeded on 13 of them. Three failures can be explained by the fact that in backward chaning, our SPARQL rule engine does not handle some recursive rules with blank nodes; the 10 other failures are due to the fact that the RIF-SPARQL dialect we have defined and implemented does not handle all the RIF-BLD equality terms.

Negative entailment test case enable to test if a rule engine can find that a given premise does not entail a given conclusion. Among the 5 such cases in the base, Corese succeeded in 4 of them. The failure on one test is due to the fact that SPARQL does not support open lists.

5 Conclusion and Ongoing Work

In this paper we have described a RIF dialect which can be translated into SPARQL. It is a subset of RIF-BLD, that we call RIF-SPARQL. Compared to RIF-BLD, RIF-SPARQL excludes universal facts, recursively defined terms and open lists.

We have implemented this dialect with the Corese semantic engine. This implementation consists of a RIF parser for both its abstract syntax and its concrete XML syntax. It builds an internal representation of RIF rules into an abstract syntax tree. This RIF AST is translated into the SPARQL AST of Corese. As a result, Corese is able to reason with RIF rules on RDF data.

A short term goal is the back translation of the SPARQL AST into the RIF AST. This will enable to produce RIF rules by writing SPARQL rules. We intend to propose our two parsers as online services. Our ongoing work deals with the comparison of RIF-BLD and SPARQL semantics and the extension of our core RIF-SPARQL dialect.

REFERENCES

[1] Renzo Angles and Claudio Gutierrez. The Expressive Power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany*, volume 5318 of *LNCS*, pages 114–129. Springer, 2008.

[2] Olivier Corby, Rose Dieng-Kuntz, and Catherine Faron-Zucker. Querying the Semantic Web with the CORESE Search Engine. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004, Valencia, Spain*, pages 705–709. IOS Press, 2004.

[3] Stephan Grimm, Uwe Keller, Holger Lausen, and Gábor Nagypál. A Reasoning Framework for Rule-Based WSML. In *Proceedings of the 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria*, volume 4519 of *LNCS*, pages 114–128. Springer, 2007.

[4] Adrian Marte. RIF4J - A reasoning Engine for RIF-BLD. Master's thesis, University of Innsbruck, 2011.

[5] Axel Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada*, pages 787–796. ACM, 2007.

[6] Axel Polleres, François Scharffe, and Roman Schindlauer. SPARQL++ for Mapping Between RDF Vocabularies. In *Proceedings of the OTM Conferences (1)*, volume 4803 of *LNCS*, pages 878–896. Springer, 2007.

[16] http://www.w3.org/2005/rules/wiki/Classification-inheritance

[17] http://www.w3.org/2005/rules/test/repository/zips/BLD_v1.22.zip