Colledge - A vision of collaborative knowledge networks

Steffen Metzger
Max Planck Institute for Informatics
Saarbrücken, Germany
smetzger@mpi-inf.mpg.de

Katja Hose
Max Planck Institute for Informatics
Saarbrücken, Germany
hose@mpi-inf.mpg.de

Ralf Schenkel
Saarland University and MPI Informatics
Saarbrücken, Germany
schenkel@mmci.uni-saarland.de

ABSTRACT

More and more semantic information has become available as RDF data recently, with the linked open data cloud as a prominent example. However, participating in the Semantic Web is cumbersome. Typically several steps are involved in using semantic knowledge. Information is first acquired, e.g. by information extraction, crowd sourcing or human experts. Then ontologies are published and distributed. Users may apply reasoning and otherwise modify their local ontology instances. However, currently these steps are treated separately and although each involves human effort, nearly no synergy effect is used and it is also mostly a one way process, e.g. user feedback hardly flows back into the main ontology version. Similarly, user cooperation is low.

While there are approaches alleviating some of these limitations, e.g. extracting information at query time, personalizing queries, and integration of user feedback, this work combines all the pieces envisioning a social knowledge network that enables collaborative knowledge generation and exchange. Each aforementioned step is seen as a particular implementation of a network node responding to knowledge queries in its own way, e.g. by extracting it, applying reasoning or asking users, and learning from knowledge exchanged with neighbours. Original knowledge as well as user feedback is distributed over the network based on similar trust and provenance mechanisms. The extended query language we call for also allows for personalization.

1. INTRODUCTION

With more and more information becoming available on the Web, the question arises how to efficiently use this global source of knowledge in an automation friendly way. The solution offered by the Semantic Web is to convert all information in machine-readable formats, such as the Resource Description Framework (RDF) and offer access over standardized interfaces, e.g. via a file or a SPARQL interface.

Although this may sound like the perfect solution, it only addresses one part of the problem, namely how to standardise information access. Other problems are addressed independently. Focusing on RDF and SPARQL, for instance, today's research already proposes strategies for efficient distributed query processing [8, 12, 13, 20, 21], source selection [1, 10, 26], reasoning [15, 18], ontology alignment [24], and semantic heterogeneity and query rewriting [7]. There are first approaches combining the steps of information extraction and query processing instead of considering them as strictly consecutive steps [11]. Furthermore, provenance provided by extraction systems can be used to retrieve further information [16]. Similarly, there is ongoing work into answering knowledge queries using crowd sourcing [6, 22]. And finally, distributed search based on peer-to-peer networks has been studied for a while as well [14, 17].

In the current Semantic Web typical tasks, e.g. information extraction, data distribution, reasoning, querying etc., are treated as independent. This leads to certain problems and inefficiencies. For instance, most components rely on human experts and demand high manual effort to be set up and operated efficiently, while synergy effects between components are rare and collaboration between different users is rather limited. For instance, humans need to identify and preselect reliable information sources, manage crowd sourcing jobs or an extraction system, set up a reasoner etc.

Users may apply reasoning to logically enhance a downloaded data set, e.g. by adding derived information or ensuring consistency or they may gather local feedback to improve the quality of the data. All modifications a user makes, however, are restricted to her local repository. Corrections, for instance, only find their way into the public release version, if a user manually contacts the original publisher and the publisher accepts the corrections.

Another problem is information freshness. In a typical scenario, information is harvested once, then stored as RDF data and published, and only rarely replaced with a new version. For instance, YAGO [23], a general purpose Wikipedia-based ontology, was last fully generated from a Wikipedia dump nearly two years ago (August 2010). While smaller updates appear irregularly every couple of months, it needs human effort to identify updated portions. Similar considerations hold for publishing data via SPARQL endpoints, although first approaches such as LiveDBpedia [2][1] try to counteract the problem of outdated data by constantly re-extracting and providing a live view onto the current data. Still, for highly dynamic data such as stock prices this might not be enough and in general, a user has no influence on the update rate and thus no guarantee how fresh the data is.

SSW'12, August 27, 2012, Istanbul, Turkey.

[1] http://live.dbpedia.org/LiveStats/

While the upcoming SPARQL version includes update functionality, these updates follow a database approach, and do not differentiate between different users and types of updates. In particular, a publisher would not want to provide external users the right to simply override their data by allowing them to execute such update queries.

In a bid to piece together several partial solutions into a general framework, this paper outlines *Colledge* (short for *Collaborative Knowledge*), a vision of a collaborative knowledge network. By modelling knowledge seekers and providers as cooperative agents acting within a social trust network our vision resembles a college, where students gain knowledge by accessing information from their teachers and information media such as books, exchange what they have learned enriched with their own modifications amongst themselves, which increases the trust they have in their knowledge, and finally may also feed back corrections and extensions to their teachers and the broader public.

In principle, our network consists of nodes that implement one or several components of today's semantic web, e.g. information caching, extraction or reasoning. Each node implements an extended query interface that allows for personalization, constraints etc. and returns a result along with meta-information such as provenance. Additionally, each node accepts incoming knowledge and may use this to optimise its own data or its processes to obtain knowledge, e.g. tuning an extraction engine.

From a user's perspective knowledge shall be distributed over this network similar to files over a file-sharing network, thus knowledge discovery is integrated into the network architecture, i.e. node discovery, and the query processing over the network. In order to automatise the selection of trusted quality information sources, a trust network is suggested and leveraged to enable a global confidence measure integrating trust into information sources and confidence of sources in their own data.

The remainder of this paper is structured as follows. While Section 2 gives more details on the current state of the art, Section 3 outlines the envisioned knowledge network, and Section 4 highlights different aspects of the system and problems to overcome. Note that an extended version of this paper (as originally submitted) is available at `http://www.mpi-inf.mpg.de/~smetzger/`.

2. THE STATE-OF-THE-ART

As an example, let us assume a startup company wants to create a website about local cinemas and the movies they show. Instead of modelling everything themselves, the company decides to use an RDF-based representation and tries to find appropriate existing ontologies.

Although the Semantic Web provides several search engines [3, 4, 9, 19] that might help locate relevant existing ontologies, the process of finding interesting sources, judging their quality and choosing the most relevant and useful ontologies involves a high degree of human interaction. Additionally, a human needs to check the accuracy of data used or trust the publisher of the data. For large publishers of general-domain information, like mainstream movies, this might be relatively simple, but it needs an expert for less well known domains, like independent French movies, or highly specialised domains, like quantum physics.

Let us assume the company has found a set of ontologies that cover most of the required information. The first obstacle is that RDF data is provided in different ways on the Web, e.g., SPARQL endpoints, RDF dumps, dereferenceable URIs, web services, etc. Some approaches offer a way to query the data on the Web without the need for a local component, e.g., SPARQL endpoints or SWSE [9], which crawls all the data and stores it into a repository that can be queried. When using such services, however, it is impossible to make corrections to the data or change it in any way, nor are guarantees for data freshness or correctness provided.

Assume, for instance, while the publishers of the original data might find it correct to place the "Lord of the Rings" movies into the horror genre, the company might disagree and change the genre accordingly. Even if the publishers accept bug fixes, they would not adapt their data according to the company's needs. Thus, for the company personalizing the ontology is a hard task if the data is not stored locally, e.g., when using remote SPARQL endpoints. If the data comes from different ontologies the company may need to merge the contained information exploiting `rdf:sameAs` links and applying techniques originating from ontology alignment, entity consolidation, and reasoning. Most of these aspects require significant effort. For instance, there are several reasoning frameworks available, which provide different benefits and have their own drawbacks. A user first needs to get a deep understanding in reasoning aspects in order to choose the system that fits her needs and then configure it accordingly.

Another issue is data freshness, e.g. the company may need to make sure that it has information on current movies as well as the schedule and number of free seats at local cinemas. Using a SPARQL endpoint view onto a regularly updated dataset may alleviate some of the update problems, although it makes integrating local adaptations only harder, but the update rate is totally up to the data publisher. While movie information will not change that often the company might want to be sure free seat informations are really up-to-date. In a SPARQL live-view there is no inbuilt possibility to negotiate the update strategy with the publishing node. Even more critical, the user might not even know how recent information retrieved is.

However, the company might decide to employ their own information extraction system instead to read information directly from the cinemas' online booking systems. Additionally, the company might provide suggestions of similar older movies on their web page along with links to buy these old movies. In order to establish movie similarity links, the company might decide to use crowd sourcing tasks. In both cases, the company would need to implement the corresponding components and integrate them. It would be hard to share the implementation as it would be designed to fit into the company's framework. Similarly sharing (partial) results would mean additional effort to publish the data and make it known to the outside world.

Even if the company decides to return some of the additional value it created in fixing and extending the base ontologies to the community that offered these so generously for free, the only way is to communicate with the publishers and explicitly report modifications.

The whole process outlined here is a tedious task done by many users over and over again because there is no straightforward way to share the result with other users.

3. ENVISIONED ARCHITECTURE

So far, in the Semantic Web there is basically no interaction or feedback between publishers and consumers. This is different in social networks where we have all become used to facebook-style "like" flags and the possibility to comment on any content. Furthermore, the recent success of cloud applications in the private sector has shown that complex processes can be made available to a broader user community if their complexity is hidden behind a common interface.

We envision a system that allows users to ignore technical details and that incorporates the interactive nature of social networks for knowledge harvesting and exchange. Similar to file sharing networks, the system is based upon a self-organizing peer-to-peer network, where each node serves as information provider and seeker while nodes are expected to cooperate to answer particular information needs. Components of the Semantic Web as e.g. extraction systems, reasoners, query engines etc., are assumed to participate in the network by implementing standardized query and communication interfaces, thus, typically complex tasks can be provided over a unified interface and be partially automatised. Additionally, steps typically seen as consecutive are interleaved and may benefit from one another as well as different users accessing the system can benefit from results retrieved for others or from modifications suggested by other users. Basically interaction with the network consists of three steps, asking a query, retrieving the results and possibly providing feedback. Each of these steps is broadcast throughout the network and each node on the way may learn from information passed to or through it. This basic layout and interaction model is outlined in Figure 1.

When a user issues a query, a single node might not be able to answer it with locally available information. Therefore, during query processing, the node might decide to apply distributed query processing techniques (cost model, indexing, statistics, query optimization, etc.) to optimize queries over multiple sources.

In contrast to existing systems, there are no restrictions on the way how (sub)queries are answered at each node, e.g., over a local triple store, using a wrapper for a relational database, involving further nodes, web services, applying a reasoning system, extracting information on-the-fly from the Web or a local corpus, as a crowd sourcing task, etc. Thus, ontological knowledge published by running a knowledge network node, similar to a SPARQL endpoint, is accessible in the same way as information acquired at query time, e.g. by a node employing information extraction or crowd sourcing methods.

Similar to social networks, the system considers and exchanges feedback on its content and its own components. A user, for instance, can give explicit or implicit feedback on the quality of a query result. This feedback is not only useful to improve the node's locally stored data, but can be shared with other nodes to improve their knowledge bases. Additionally nodes can employ user feedback to improve their knowledge acquisition methods, e.g. to tune an information extraction system or to adjust the trust in a particular worker when a crowd sourcing approach is used.

Also similar to social networks, the system needs to consider the concept of "friends" or trust, respectively. Not all the sources can be trusted, neither in the data they provide nor in their feedback. Hence, nodes need to provide a trust evaluation and a trust network needs to be established.

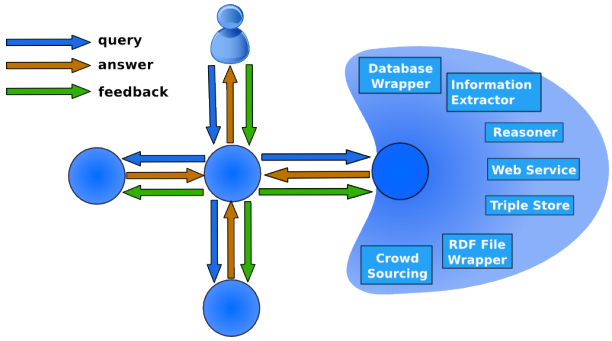

Figure 1: Network Architecture Outline

Sources: User clipart symbol from OpenCliparts.org (public domain)

Thus, when answering queries, provenance is an essential aspect. Provenance information can be used, first, to establish whether information received is trustworthy, and second to find original source nodes that provide interesting information to the network. This way, high quality nodes can be found and be queried directly in the future. However, for basic network exploration, nodes will also need to provide meta-information about themselves, similar to the voID ontology descriptions used for linked open data[2] allowing other nodes to decide whether they might be interested in a node's information services. Additionally nodes can offer update subscriptions to deal with dynamic data.

From a user's perspective, the system is similar to a social file sharing network, where all users can participate in as soon as they know any node that already participates. Ideally, the user formulates a query and the system does everything automatically – it identifies relevant sources, rewrites and extends the query, considers mappings between ontologies, applies reasoning to derive further information etc.

The system also allows a user to restrict and personalize a query based on her own beliefs, e.g., if aliens exist or not. On the other hand, a user might want to test her beliefs by issuing a consistency query, that explicitly aims to find indications that contradict her basic beliefs. Therefore, the system also incorporates reasoning as well as reasoning nodes so that a user does not need to find, install, and configure a reasoning framework herself.

In order to help the user formulate structured queries, the system can make use of approaches that map natural language queries into structured queries. In addition, the system is interactive, i.e., in case of ambiguities with respect to query interpretation, the system will ask the user and learn from her responses.

Let us consider the application scenario from Section 2, where a company builds a website about local cinemas, providing information such as their location, which movies are shown and how many places are left for a particular showing.

With Colledge, the company first sets up a node within the network or accesses an existing one. This will require some effort the first time it is done, but that node can be reused for multiple applications.

Once the node has run an initialization phase to discover its neighbourhood, the company needs to manually find out the best way to canonically formulate their queries and then train the node for typical queries. However, instead of reading ontology descriptions and searching for fitting ontologies,

[2]http://www.w3.org/TR/void/

a human simply issues natural language queries and decides whether the results fit her expectations. From the results she can also derive typical entity and relation names to use in the automatic query generation later employed by the web application. For instance, a developer who wants to program a method to retrieve all horror movies currently shown in "Steve's Cinema" might simply issue a natural language query for "Steve's Cinema shows what horror movies?"which might be translated into a query with the SPARQL triple patterns (`cs:StevesCinema,cs:shows,?m`).(`?m,rdf:type,imdb:horror`). The system might then come up with triples of the form (`cs:StevesCinema,cs:shows,imdb:LotR`). (`imdb:LotR,rdf:type,imdb:horror`) at which point the developer can directly use canonic expressions to formulate the query in the future. Note that the possible complexity of the natural language interface depends mainly on the capabilities of the translation module, we are using a simple example here.

Figure 2 outlines a possible distribution of the example query through the network. Note, that several types of nodes are involved (see Section 4.2). The query enters the network at Q, a simple relay node that separates the query q into its two components $q1$ and $q2$. An aggregator relay node then retrieves answers for $q1$ from two nodes offering information on movies, while $q2$ is processed by a reasoning node, which reduces $q2$ to $q3$ and $q4$, retrieved from an aggregator memory node AG. This aggregator uses amongst others an extraction node (SC) which regularly parses the website of Steve's Cinema and informs AG about updates. Alternatively SC could not offer an update subscription, but extract only at query time. However, in this example the data should change at predictable times, thus a fixed extraction schedule may be more plausible.

Once the answer is received, the company might flag the answer or a particular triple as invalid, e.g. since the company does not consider "Lord of the Rings" an horror movie. This feedback will be sent back to all nodes involved in retrieving the query answer and they may decide to accept or ignore this opinion. If the source node accepts the feedback as valid, the company's query node will never see the answer again. Even if the feedback is not initially accepted, the query node will always filter this particular answer out, either by placing it as a query constraint along with any such query or by filtering the answers by comparing them with the local user provided knowledge. In the next section we shall discuss some aspects in more detail.

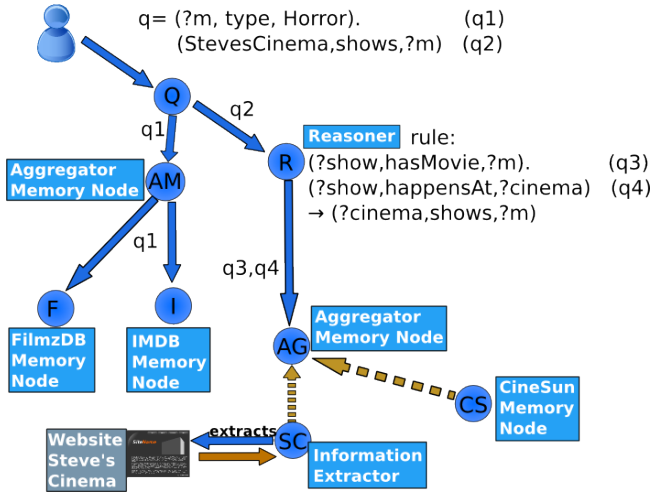

Figure 2: Query Example

Sources: User and website symbol from OpenCliparts.org (public domain)

4. COLLEDGE CORE ASPECTS

In the following, we outline some of the core aspects considered by the envisioned framework and identify some of the main challenges that remain to be solved.

4.1 Data and Query Model

While the proposed framework could be based on any data model or query language, we focus on RDF and SPARQL as these are the de facto standards used in the Semantic Web. Thus, information is represented as RDF triples, i.e., $s =$(`subject,relation,object`) expresses that a specified `relation` holds between `subject` and `object`. In the following, we refer to triples expressing certain information as *facts* and to triples expressing uncertain information as *statements*. Statements usually come with a *confidence value* $conf(s)$ $\in [0, 1]$ indicating how likely the statement expresses a true piece of information – facts are statements with $conf(s) = 1$.

With respect to SPARQL queries, we will often neglect bindings of variables in the select clause that actually represent the result because from a system's perspective, we are more interested in the triples that led to the query result with attached information about confidence and provenance. Thus, we define the set of *result triples* as the set R of triple subsets r producing the complete set of answers (bindings) for the query q over a knowledge base K:

$$R := \{r | r \text{ produces a valid answer for } q\} \quad (1)$$

with $r := \{s_1, \ldots, s_n\} \subseteq K$. For a query over the network, K represents the set of all statements available in the network.

In distributed systems, the terms *query coordinator* or *query origin* denote the node in the network that issues a query, i.e., the node at which a user (or any other software running on the node) initiates the query processing. In the following, we will use these terms to denote which node a query originates from and therefore where the final result to a query has to be available.

4.2 Knowledge Network and Node Classification

Formally, a knowledge network can be defined as a set of knowledge nodes N connected by a set of directed edges E, where each edge from node n_1 to node n_2 indicates that n_1 is aware of n_2's existence and knows how to access n_2's data. Each node $n \in N$ needs to implement a query and a feedback interface so that it can receive and answer queries and learn from feedback.

The query interface can be implemented on top of an extended SPARQL endpoint or in principle by any other kind of service generating RDF data. Thus, from a technical point of view, we distinguish three main categories of nodes: (i) *relay nodes*, (ii) *memory nodes*, and (iii) *generation nodes*.

Relay nodes maintain indexes and operate as query re-routers.

Memory nodes provide access to stored semantic data, e.g. by masking an RDF triple store, reading an RDF file or wrapping another structured format (e.g. a database).

Generation nodes generate information on-the-fly. For the time being, we envision nodes applying information extraction, crowd sourcing, reasoning nodes, or sensor nodes. While information extraction nodes employ an information extraction system to generate information from source documents (at least partially) at query time, crowd sourcing nodes use crowd sourcing tasks to generate RDF triples [6, 22]. Reasoning nodes use logical rules to (recursively) derive knowledge from existing data during runtime.

In practice, a node may belong to multiple categories.

4.3 Provenance and Trust

To determine the quality in terms of soundness and trustworthiness of query results in a distributed setup, an important piece of meta information is how the results were derived from existing knowledge and where this knowledge originates from. Thus, during query processing additional meta information needs to be exchanged.

To formally define the origin of data as part of the meta data, we use the notion of a *knowledge source* (Definition 4.1).

DEFINITION 4.1 (KNOWLEDGE SOURCE, PRIMARY SOURCE). *A* knowledge source *of a statement s provides a representation of s. A knowledge source of a statement not referencing another knowledge source as its origin is called a* primary source *of the statement.*

To identify the origin of information, nodes answering a (sub)query provide provenance information on their local results. In the simplest case, provenance information for a result triple resembles a chain:

DEFINITION 4.2 (SIMPLE PROVENANCE CHAIN). *Given a query q and a node n that produces statement s as triple result to q, a* simple provenance chain *is a sequence of knowledge sources $n, n_1, \ldots, n_m$ reflecting the nodes via which statement s has been retrieved.*

Ignoring reasoning for the moment, for each piece of information, there is at least one simple provenance chain from a *primary source* to its final recipient. The trustworthiness of a piece of information is usually based on such provenance with special regard to primary sources. Typically, confidence in a statement s

- increases the more *provenance chains* are based on different primary sources,
- increases the more different provenance chains are based on the same primary source ps, but the trust held in ps can never be exceeded,
- increases the shorter the path to a primary source is.

Consider the example in Figure 2 and sub-query $q1$. Assume that node I returns a response triple s =(`LotR`,`type`, `Horror`). Then Q, AM, I is a simple provenance chain and I is the primary source for s. Q might now either trust the information chain or verify the data directly at the (primary) source, in this case at I.

Either way, at some point, the *confidence* in s needs to be estimated. This will depend on the *trust* in the knowledge sources of the remaining provenance chain. Basically, in order to believe a statement s to be true, one needs to trust each node along the way to be honest. Thus given a simple provenance chain $q, n_1, ..., n_w$ and a trust function t on knowledge sources, we can estimate the confidence $conf(s)$ in s by

$$conf(s) = \prod_{i \in \{1,..,w\}} t(n_i) \tag{2}$$

Knowledge sources might also be aware that some information they provide is less certain. An information extraction system might for instance know that its extraction methods are less reliable for some relations or particular statements. Thus, a node i might provide its own confidence estimation $conf_i(s)$, indicating how confident it is to have understood or measured the information correctly, which can also be included in the overall confidence product.

Note that this is only a simplified outline for basic cases. In practice, provenance information for a query answer resembles a directed acyclic graph where nodes represent logical operators. Consider, for instance, s being retrieved not only from I but also from F via AM. Then, at AM we can believe in s if we trust I or F. Similarly, reasoning nodes can reduce a statement s by applying logical rules to a set of pre-condition statements $c_1, ..., c_n$, resulting in s being true iff we believe in $c_1, ..., c_n$, thus introducing additional logical nodes in the provenance graph.

There are methods in probabilistic reasoning to derive a probabilistic confidence from such provenance graphs [25]. While the worst case runtime for such methods can be exponential, they can still be applied in real-time [15] as provenance graphs usually have a limited size.

These trust computations can either be done at the query origin or a node might delegate trust estimations by trusting the confidence estimations received from neighbours along with query answers. Recursively applied, confidence is computed stepwise based on the local trust function at each node the query passes. This approach also has the advantage that it allows for more efficient optimization within the network. For instance, partial results can be ordered by local confidence and only the top results be returned. The final setting may apply a mix of these approaches which may be depending on negotiations between a querying and the answering node. After all, some nodes might not be willing or able to provide provenance information but might be able to give a confidence estimation or the other way around.

4.4 Generation Nodes & Data Freshness

Our knowledge network includes nodes that can acquire information at query time to deal with changing information. Important examples include information extraction and crowd sourcing, but also sensor and reasoning nodes.

We assume that only a minority of all statements change over time and their change frequency varies heavily, e.g. consider a change in temperature versus the change of a movie theatre's schedule. Thus, it is not reasonable to update information more often than it actually changes. On the other hand, a generation node might have limited update capabilities. If we imagine a sensor node measuring the temperature at the North Pole, taking a particular measurement is a matter of less than a millisecond. An extraction node re-parsing several web pages or a crowd sourcing node, for instance, retrieving crowd opinions on the quality of a current movie either by graded rating or a 500 word review – in Gaelic, may take seconds, minutes or even longer. Thus, a node may be limited physically to a maximal update frequency. Additionally, limitations like available computation

power and parallel queries can add a dynamic limit on the actual update frequency. Depending on the actual information need, information freshness requirements may also be variable. A querying node might be satisfied if the information retrieved contains all updates to a certain time point or contains on average only a certain percentage of outdated triples. For instance, if the user needs to estimate whether to bring a bikini or a snow suit for her holidays, it is tolerable if the temperature at her holiday destination was acquired a few days ago. Similarly, if calculating the average number of movie ticket sales over a weekend to calculate whether a movie achieved a box office record, a margin of error might be acceptable in a first estimation, thus a certain percentage of visitor numbers being not up-to-date would be acceptable.

Thus, in our model a querying node can attach a hard or soft data freshness constraint to its query. In the soft case, the answering node can decide to answer although it cannot satisfy the constraint, but a hard constraint indicates the query coordinator will throw away any result not within the constraints anyway, thus it can spare such answers.

However, while a node may not be able to always provide the most up-to-date information, the least it needs to provide is meta-information on how fresh its information is.

Finally, aggregator nodes can divert traffic from primary source nodes by caching their knowledge on particular knowledge domains, e.g. on movies. In the example in Figure 2 the nodes AM and AG are examples of such aggregator nodes. An aggregator node may decide to subscribe to an aggregated node's updates of certain information types, if the aggregated node offers such an update stream. A typical example where this makes sense are informations that change infrequently, such as a movie schedule. In the example in Figure 2 AG has subscribed to update streams from CS and SC and thus, never needs to ask for information covered by these subscriptions.

4.5 Personalization

While there is often great emphasis on constructing consistent knowledge spaces, we explicitly suggest the creation of an inconsistent knowledge space within the network. The knowledge network should represent human understandings of the world, which are often enough inconsistent, especially when beliefs held by different people are mixed. Thus, to retrieve meaningful answers, the system may need to decide at query time which particular statements to accept as base facts. This decision should depend on the beliefs held by the user. These can be explicitly stated at query time as query constraints or gathered over time from user feedback (see Section 4.7). For instance, a user query may be extended by the statement (`LotR`,`type`,`Horror`) to indicate that the user thinks the movie "Lord of the Rings" is a horror movie, and thus the query result for movie suggestions to see with her young nephew should not contain it. Again, such query constraints could be applied as a filter once all query answers reach the query node along with full provenance information, but in most cases it would be more efficient to provide such constraints as part of the query, such that results in conflict with the query constraints are filtered out locally within the network.

Note that the full power of query constraints is only used when reasoning is employed. Basically either the query node first derives as many consequences from any user specific query constraints and adds these as well or individual nodes need to employ reasoning, possibly via a reasoning node in order to include implied consequences derived from the constraints specified.

4.6 Query Processing

The first step of processing a query that cannot be answered locally is to locate nodes with matching knowledge. In today's Linked Open Data (LOD) setup, the user needs to manually identify candidate nodes and add them to their federation [21]. While such a manual setup should still be possible to add known trustworthy nodes, we envision that knowledge nodes automatically discover other nodes using techniques similar to peer-to-peer networks.

Once potential sources are identified, the query processor can simply split up the query and ask every known node, similar to the current approach for LOD. In Figure 2 the query node first splits the original query q into its two sub-queries $q1$ and $q2$, solves the sub-queries and then joins the results. Each query executed yields information about which nodes can provide which kind of information, including information about transitive neighbours and especially their primary sources. Thus, a node can iteratively learn more and more about nodes in the network. Unlike current LOD settings, however, these nodes can be of very different kind, so the quality of information provided by a node can vary and needs to be estimated. This allows for ranking nodes and asking nodes according to the ranking in batches until the result is satisfying. Typically, memory nodes will provide results faster than generation nodes, but there may also be differences amongst generation nodes. This node efficiency can be learnt iteratively just as the trust distribution. To speed up the learning process for both properties, i.e., information availability and query processing efficiency, we require that every node provides general meta-information, such as its type and the knowledge domain it covers, similar to the voID descriptions used in the LOD world [1].

Given both user-defined goals and per-node estimations for quality and efficiency, a cost model is needed to determine optimal query executions. Existing work on integrating information extraction on-the-fly into SQL [5, 11] often includes such cost models, but they need to be extended with per statement confidence. Additionally, query execution in our network is distributed and thus it could leverage existing distributed SPARQL approaches [21]. As query run-time at different nodes may vary largely, we envision an incremental processing strategy, reporting initial results early while continuing evaluation in the background, such that they are available when a user asks for more or more convincing results. To alleviate this, a querying node can attach time or quality constraints to a query, such that a queried node can either tune its execution plan or directly refuse execution.

An important aspect not considered by existing systems are the different update rates of different types of information. Depending on how often the information changes and how up-to-date the information needs to be, it may not always be necessary to re-extract. This balance between data dynamic and user constraints on data freshness needs to be integrated into the cost model as well.

4.7 Feedback

User feedback in our setting can be given either explicitly by pointing out that particular statements are not true according to the user's belief system, or implicitly, e.g. by

learning from query constraints used by the user, by results she usually accepts and vice versa by learning from results that seem not to satisfy the user, such that she asks for more. Thus, by constantly harvesting a user's feedback and query constraints, a knowledge node may generate user belief statistics, i.e. a set of statements a user finds valid or invalid associated with confidence values indicating how certain the node is that the user holds this belief. The system can then re-use this information for personalization and to improve in several ways:

1. It may include statements a user beliefs in as default query constraints
2. It may use statements a user beliefs in as answers for incoming queries, citing the user as source
3. It may adjust the (user specific) trust function for a source whenever the user accepts or rejects statements that stem from this source
4. If the node has an information extraction component it may adapt confidence estimations for the method used to extract the statement

While feedback is typically collected after a query has been answered, the system might also directly ask the user during query execution to clarify the query or her assumptions about the knowledge domain. For instance the system could

- try to clarify disambiguation problems to understand the query, e.g. asking which "Einstein" the user meant, the famous scientist or the dog from the "Back to the Future" movies
- ask whether it can extend or restrict the query in order to improve the result quality or the performance, e.g. by replacing an entity type like `GermanActor` by a more general type like `Actor` or vice versa
- try to clarify some basic beliefs that are decisive for the query execution, e.g. whether the user accepts that mankind has visited the moon or not

While such interaction may make a huge difference in terms of execution cost, when the right questions are asked, the number of user questions needs to be limited such that the user does not get annoyed by answering her own query. Hence this could also be integrated into the cost model with an additional user discomfort cost and a maximum discomfort threshold depending on the user.

4.8 Collaboration

Arguably the main difference we would like to encourage is that of (semi-)automatic collaboration. While today, knowledge publishers can only gain from one another if they manually talk to each other or copy parts of another ontology (in case they do not agree with parts of its modelling), which might be considered plagiarism by the original authors, we want to explicitly encourage and support information exchange. In particular this means nodes in our network architecture should collaborate and exchange:

- ontological knowledge relevant for query answering
- provenance information and thus, knowledge about their information sources
- index information recommending expert nodes
- user feedback by forwarding feedback to relevant nodes
- meta information on good extraction, crowd sourcing or sensor settings

We have previously discussed the typical answering of queries as well as how users can generate feedback and how it can be used locally to improve a node's performance. However, once a query is answered and feedback from a user gathered, it is the responsibility of the query node to try pay back some value to the nodes that helped to answer the query. Thus, user feedback should be forwarded to all nodes involved in the query answering, more precisely to all nodes potentially affected by the feedback. For instance, if user feedback claims a certain statement to be false, feedback shall be sent to nodes that were responsible in producing that part of the answer. However, the query node may decide to contact additional nodes, if it thinks these may also benefit due to its semantic node index. Additionally a query node may have contacted different nodes and thus it might have received different, potentially disagreeing results. It should also inform nodes that produced disagreeing results about the opinion of its counterpart. This also holds for any node that issued a sub-query.

Consider the example in Figure 2 and assume the query results amongst others in the IMDB node I returning the triple $s = (\texttt{LotR}, \texttt{type}, \texttt{Horror})$ towards the aggregator node AM, while F provides a triple $\overline{s} = (\texttt{LotR}, \texttt{not\_type}, \texttt{Horror})$ indicating the opposite. Since I enjoys more trust, at the query node, s will be retrieved and be part of a query answer. If now a user refutes this part of the answer, Q will forward the feedback to AM, which will forward it to I and M, which may update their knowledge. Q and AM may also adapt their trust function for I and M.

Similarly nodes can exchange index information, which can be especially helpful for initial node discovery. While we already suggested that each node provides an interface for self-description, this could be extended to also offer descriptions on other nodes, based on the index information available. This way, a node new to the network could ask initially known neighbours explicitly for the best expert nodes to be asked about certain kinds of information, thus shortcutting the discovery process further.

Additionally, nodes may gather and exchange user belief statistics (if users agree) in order to generate user clusters to improve indexing and query processing as well as try to establish belief worlds, i.e. sets of statements agreed on by a substantial amount of users. This can help again in improving result quality by employing automatic query rewriting techniques, e.g. adding more constraints from the belief world a user belongs to. If user statistics can be exchanged, this could even lead to social interaction amongst users that share similar beliefs, which might also simply mean they are interested in similar knowledge domains.

Finally, the network may collaboratively solve larger tasks like ontology matching [24] and entity consolidation in a distributed way. For instance, during query processing reasoning nodes might logically derive new `sameAs` links. This information can be distributed through the network as part of a query answer or in a dedicated exchange step. The more nodes agree on a `sameAs` statement, the more likely it will establish itself as a fact in large portions of the network and thus become accepted truth. This can help in

ontology matching as well as entity consolidation. There could also be dedicated nodes coordinating such global endeavours, e.g. by collecting `sameAs` candidates and spreading candidates accepted by enough reasoning nodes deliberately within the network.

5. CONCLUSIONS AND FUTURE WORK

In this paper we envision a distributed self-organizing social knowledge network to enable collaborative knowledge generation and management. We give an overview over several important research areas related to semantic knowledge generation and management and outline a way to combine these issues in a shared architecture. Our envisioned network aims to lower the participation threshold and encourage a higher level of knowledge exchange and interactivity. While trying to ease access and still give users as many personalization freedoms as possible, participating nodes are also free to decide about their own policy.

While we sketch a rough outline of a knowledge network of the future, in each aspect we discussed there remain open problems to overcome before such a network can be realized. Among the major obstacles that need to be tackled towards such a knowledge network, we leave one aspect relatively untouched, namely that of security. While a trust network might protect against some forms of knowledge oriented attacks, other forms of network attacks, like denial of service attacks and data redundancy are not addressed at all.

6. REFERENCES

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets – On the Design and Usage of voiD, the Vocabulary of Interlinked Datasets. In *LDOW*, 2009.

[2] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semant.*, 7(3):154–165, Sept. 2009.

[3] G. Cheng and Y. Qu. Searching Linked Objects with Falcons: Approach, Implementation and Evaluation. *Int. J. Semantic Web Inf. Syst.*, 5(3):49–70, 2009.

[4] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM*, pages 652–659, 2004.

[5] A. El-Helw, M. H. Farid, and I. F. Ilyas. Just-in-time information extraction using extraction views. In *SIGMOD*, pages 613–616, 2012.

[6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[7] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View selection in semantic web databases. *PVLDB*, 5(2):97–108, 2011.

[8] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *ESWC (1)*, pages 154–169, 2011.

[9] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and browsing linked data with SWSE: The semantic web search engine. *J. Web Sem.*, 9(4):365–401, 2011.

[10] K. Hose and R. Schenkel. Towards Benefit-Based RDF Source Selection for SPARQL Queries. In *Semantic Web Information Management (SWIM)*, Scottsdale, AZ, 2012.

[11] A. Jain, P. G. Ipeirotis, A. Doan, and L. Gravano. Join optimization of information extraction output: Quality matters! In *ICDE*, pages 186–197, 2009.

[12] G. Ladwig and T. Tran. SIHJoin: Querying remote and local linked data. In *ESWC (1)*, pages 139–153, 2011.

[13] A. Langegger, W. Wöß, and M. Blöchl. A semantic web middleware for virtual data integration on the web. In *ESWC*, pages 493–507, 2008.

[14] J. Lu and J. Callan. Content-Based Peer-to-Peer Network Overlay for Full-Text Federated Search. In D. Evans, S. Furui, and C. Soulé-Dupuy, editors, *RIAO*. CID, 2007.

[15] T. Meiser, M. Dylla, and M. Theobald. Interactive reasoning in uncertain RDF knowledge bases. In *CIKM*, pages 2557–2560, 2011.

[16] S. Metzger, S. Elbassuoni, K. Hose, and R. Schenkel. S3K: Seeking Statement-Supporting top-K Witnesses. In *CIKM*, Glasgow, UK, 2011.

[17] S. Michel, P. Triantafillou, and G. Weikum. MINERVA∞: A Scalable Efficient Peer-to-Peer Search Engine. In *USENIX 2005*, volume 3790 of *Lecture Notes in Computer Science*, pages 60–81, Grenoble, France, 2005. Springer.

[18] B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[19] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, 3:37–52, 2008.

[20] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538, 2008.

[21] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *ISWC*, pages 601–616, 2011.

[22] J. Selke, C. Lofi, and W.-T. Balke. Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. *PVLDB*, 5(6):538–549, 2012.

[23] F. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge - unifying WordNet and Wikipedia. In *WWW*, pages 697–706, 2007.

[24] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB*, 5(3):157–168, 2011.

[25] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[26] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over linked data. *World Wide Web*, 14(5-6):495–544, 2011.