Link Traversal and Reasoning in Dynamic Linked Data Knowledge Bases

Andreas Harth

Institut für Angewandte Informatik und Formale
Beschreibungsverfahren (AIFB)
Karlsruher Institut für Technologie (KIT)

Habilitationsschrift (kumulativ) zur Erlangung der venia legendi

Link Traversal and Reasoning in Dynamic Linked Data Knowledge Bases

Andreas Harth

December 21, 2015

List of Publications

[P1] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data Summaries for On-demand Queries over Linked Data. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 411–420. 2010.

[P2] Sebastian Speiser and Andreas Harth. Integrating Linked Data and Services with Linked Data Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, pages 170–184. 2011.

[P3] Andreas Harth and Sebastian Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*. 2012.

[P4] Anisa Rula, Matteo Palmonari, Andreas Harth, Steffen Stadtmüller, and Andrea Maurino. On the Diversity and Availability of Temporal Information in Linked Open Data. In *Proceedings of the 11th International Semantic Web Conference (ISWC)*, pages 492–507. 2012.

[P5] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 1225–1236. 2013.

[P6] Andreas Harth and Steffen Stadtmüller. Parallel Processing of Rule-based Programs on Linked Data. 2015. Under review.

Contents

1 Introduction

1.1 Motivation

Computer technology has led to a rapid growth of available data: enormous amounts of data are created by sensors or people interacting with devices. Most of these sensors and devices are joined together via communication networks. Communication requires a channel to exchange messages and a shared language between sender and receiver to encode these messages.

The World Wide Web provides a widely-used technology stack for communication [7]. The web is currently evolving from an infrastructure for publishing and accessing documents to an infrastructure for publishing and accessing data. Linked Data [6], building on web technologies combined with a graph-structured data format and decentralised linking, provides means to cover both communication protocol and representation of the semantics of messages. While previous approaches handled communication and machine-interpretable semantics separately, in our work we bring together network access to interlinked data with logic-based knowledge representation and logic programming approaches.

Providing a unified approach for both data access and data processing brings benefits in scenarios that require the integration of data from multiple sources or the interoperation between disparate systems. The approach we follow is geared towards scale, that is, we assume that there is a large number of data sources and systems under diverse ownership. Our approach builds on hyperlinks for connecting different sources and systems together in a decentralised way. Such a decentralised approach has been successful with the web of documents. Now, we can see the beginnings of a web of data based on the Linked Data principles. For example, in the areas of digital libraries and life sciences, there are first systems deployed which provide integrated access to a fixed number of data sources based on Linked Data [5, 19]. So far, however, none of these systems harness the power of arbitrary hyperlinks put in by a large number of people in a decentralised manner, as in the case of the hypertext web.

Our hypothesis is that hyperlinks can be used to provide integrated access to the web of data. To test the hypothesis, we have to provide methods operating on the web of data that use similar techniques for link traversal to those methods we know from web browsing. Given that the content on the web of data is represented in graph-structured form in the Resource Description Framework (RDF) [46], users should be able to access the data using

declarative queries over the data. In addition, users should be able to automate link traversal with appropriate languages for encoding link traversal behaviour.

It is important to note that we do not talk about data integration scenarios where a predetermined set of files are loaded into a local repository and then queried, as is common in most data integration systems [21]. In other words, the usual approach for data access and integration today is to specify one (or a few) sources, create mappings on the schema and instance level, download the entire data from these sources, and then try to provide integrated query capabilities over the local data "warehouse". Rather, we assume a web environment, where content authors provide additional links from their data to other data in a completely decentralised manner. Further, instead of running the integration step infrequently (once a day, say), we would like to be able to run the integration step at regular intervals. In the web scenario, the data integration system should take these links into account, and actively dereference data from linked sources during query processing. Then, data providers can potentially produce a completely different result of the data integration process simply by adding hyperlinks to their sources. In case the data integration system hits the source in question, the system follows links from that source during runtime, and then again links from the linked source, and so on. Such an on-demand approach is beneficial in applications that require access to fresh data, and naturally extends to applications that change a data item in a remote system (to book a concert ticket, for example).

In the web of documents, users access documents via a user agent (for example, web browsers). These documents, connected via hyperlinks, are published by a large number of people. Now, a browsing session works as follows. Users start from a document, and just follow hyperlinks prepared by the document authors, until the users have satisfied their information need (or abandon their browsing session).

Our assumption is that in a web of data, the process should work analogously: user agents start from a document containing data, and just follow hyperlinks to other documents containing data. With access to data instead of hypertext web pages, users should be able to automate the link traversal activity, and the user agents operating on data should use semantics provided by the sources to be able to integrate the data.

The interfaces specified by web architecture are domain-independent, and can used to publish and access both hypertext documents and data. Further, these interfaces can be used to provide access to internet-connected

physical devices (the so-called "Internet of Things"), leading to a "web of things"[1]. In a web of things, provided there are hyperlinks, the browsing process still works. In such scenarios, users require not only access to sensors, but also the ability to incorporate effectors; manipulating data becomes as important as accessing data[2]. If we assume an interface that allows for agents to change the state of resources, hyperlinks and semantics can be also used to interconnect devices and components from disparate systems.

1.2 Information Ecosystems

As the amount of data increases, the number of systems providing and processing data increases as well, leading to complex information ecosystems with massive amounts of data, a very large number of data sources and systems operated and used by a multitude of stakeholders [24].

We assume a decentralised system for sharing and accessing information. While centralised components simplify coordination, centralised services require an organisation to operate the service, which leads to a power structure and a single point of failure. On the other hand, there is increased technical complexity when building systems without centralised components[3]. A single team responsible for integrating a small number of sources may be quickly achieving results with imperative programs and glue code. But if we assume a very large group of participants distributed across the globe, and we want to achieve integrated access to their data and interoperation between their systems, imperative programs and glue code may prove unsustainable. Rather, it makes sense to do the up-front investment of providing a common abstraction to build on, to ease integration and interoperation. In the case of the web, that abstraction is based on resources and hyperlinking.

In other words, in large distributed systems we need a flexible and extensible approach for data access, integration and manipulation. At the same time, a constrained minimal interface, which is easy to learn and simple to implement, facilitates widespread adoption.

Consistency is a recurring challenge in open systems of interconnected and interdependent computers [27]. Because many people contribute to a

[1] `http://www.w3.org/WoT/`

[2] While there have been ways to update data in HTML (via web forms) and Linked Data (via RESTful Linked Data APIs), in the web of things the ability to change a resource state is crucial.

[3] There are a few centralised components in the architecture of the internet. For example, the assignment of IP addresses is centralised, and so is the Domain Name System (DNS) - albeit those are organised in a hierarchical fashion.

potentially huge distributed systems, mistakes occur and errors have to be dealt with. On the internet, the so-called robustness principle applies: "be conservative in what you do, be liberal in what you accept from others"[4]. In link traversal systems, we might have to accept arbitrary data from the sources reachable via links. To be able to process data with hitherto unknown identifiers, we need semantics that describe the data items.

The integration of arbitrary data from the web represents a formidable challenge. However, the potential gain in functionality is tremendous. Based on the available data, data consumers can easily pick and choose the appropriate data for a specific application. Access to a large quantity of data becomes a quality in itself [22]. Following hyperlinks and dealing with the semantics provides the basis for integrated access to vast amounts of data from massive numbers of web sources.

On the web, the Hypertext Transfer Protocol (HTTP) is the single protocol for universal data transfer. Instead of allowing arbitrary operations, HTTP defines a minimal set of operations to create, read, update and delete resources (CRUD)[5]. HTTP is robust in the face of unreliable networks or changing network connections, with stateless request/response communication methods, instead of persistent connections or sessions on the socket layer as with Telnet or the Simple Mail Transfer Protocol (SMTP). In HTTP, each request is self-contained; that is, each request contains all information necessary for the server to process the request independently of previous requests, which facilitates load balancing and caching.

The focus of web architecture on a minimal set of operations in a request/response paradigm is a constraint. However, the restriction that the architecture imposes provides a fixed structure (and less degrees of freedom). Such a fixed structure makes it easier for data publishers and consumers to operate in.

Approaches that take a resource-centric view based on the core web standards URI and HTTP with a limited set of operations (often subsumed under the acronym REST, short for Representational State Transfer) are now more popular than "enterprise" standards that allow for the definition of

[4]Also called Postel principle, after Jon Postel, long-time editor of the Request for Comments documents, among them RFC 761, which specifies the Transmission Control Protocol (TCP) `https://tools.ietf.org/html/rfc761#page-13`.

[5]HTTP is very carefully extended to cater for updates and streaming data. RFC 5789 introduces a PATCH operation to be able to specify updates without sending the entire representation, RFC 7252 introduces the Constrained Application Access Protocol (CoAP), a lightweight version of HTTP designed for accessing sensor data in web of things scenarios, and adds an OBSERVE operation.

arbitrary operations in the style of remote procedure calls [42]. On the web, REST APIs have overtaken APIs based on enterprise SOAP/WS-* technologies[6]. Uniform Resource Identifiers (URIs) and HTTP are ubiquitous, HTTP APIs are commonplace, programmers are familiar with the protocol stack, and tools and libraries for HTTP are mature and widely available. Newly designed interfaces often build on HTTP rather than inventing a new wire format for the protocol. Thus, web architecture – the use of URIs to identify resources, and HTTP as transport protocol – is the dominant way of accessing and manipulating data in networked systems.

Consequently, we assume as basis for our work an ecosystem consisting of:

- a network stack for communication;
- global identifiers for naming and referencing resources (URIs);
- a communication protocol for accessing (reading) and manipulating (writing, creating, deleting) state representations of URI-identified resources (HTTP);
- a graph-structured data format (RDF) and knowledge representation languages, viz. RDF Schema (RDFS) and subsets of the Web Ontology Language (OWL);
- a community of authors providing data including links to other data, developers providing software, and people who run and operate the infrastructure providing access to data and services.

1.3 Challenges

The overarching challenge we tackle relates to the fact that our goal is the integration of heterogeneous data and the interoperation between diverse systems on a very large scale. Web architecture provides the technologies for a minimal but universal interface to access and manipulate a broad variety of content and functionality in various systems. Further, web architecture heavily relies on hyperlinks for resource discovery.

Linked Data adds RDF as graph-structured data format. Many sources use in addition lightweight knowledge representation languages (RDFS, parts of OWL). In experiments and prototype systems, we leverage the vast amount

[6] ProgrammableWeb (`http://www.programmableweb.com/protocol-api`) lists 8,905 REST APIs and 2,432 SOAP APIs as of November 20, 2015.

of data that has been made available by the community of Linked Data enthusiasts. While many Linked Data sources are currently read-only, there are standardisation efforts for providing write access to systems in form of the Linked Data Platform recommendation [48].

To be able to use the available Linked Data on the web and answer an information need, we require methods to navigate the large interlinked data graph. In other words, we require methods to specify link traversal. The information need can be expressed as a query Q over given datasets or systems D, while D is distributed, hyperlinked and not completely known a priori. Our methods assume an entry point: an initial source $d_0 \in D$, or a subset of sources $D_0 \subseteq D$, similar to how a web browsing session starts. From then on, our method relies on following links to access more data. Such a method is in contrast to traditional data integration systems, where there are few, large sources that are predetermined. In our case, D consists of a massive number of small sources that are interlinked. We thus require a means to specify what links should be traversed, and actual systems that carry out the traversal. Further, as the users do not know D when formulating their information need, the users might express the query in a different vocabulary to the vocabulary in which the data is encoded in. Thus, we need ways to reconcile heterogeneity in the data.

We identify the following challenges in the described ecosystem:

- C1: A challenge refers to methods and algorithms to access and query the distributed data. Are there ways to deal efficiently with the large number of small sources that constitute Linked Data? How can we provide read-access to data that is created dynamically?
- C2: A hyperlinked environment, which can actually be infinite, directly raises the question of completeness: how does a system know when to finish accessing sources? How can a system make sure that it accesses all data required for answering a given query?
- C3: Data on the web is not static but dynamic and thus changes over time. The modelling of temporal information is different across sources on the web. What are the characteristics and the quality of temporal data on the web?
- C4: Given that the data is created by many different authors that do not coordinate, we require more fine-grained specifications of link traversal than those that can be encoded in a query. How can we specify link traversal with concise, declarative means?

- C5: While the ability to answer queries over large amounts of integrated data from multiple sources is a first step, for full interoperation between systems we require the ability to include effectors. How can we create systems that not only access, but also manipulate the state of systems in hyperlinked environments?

- C6: Given the large data volumes, we require scalable algorithms and methods for data access, query processing and reasoning. Because data access is IO-bound, and data processing is CPU-bound, what are methods to balance both type of tasks in a parallel setting to improve performance?

1.4 Contributions

Our approach takes the architecture of the document web as starting point and transplants that architecture to the realm of data. Compared to more traditional data management methods, our approach builds on hyperlinks for data discovery and thus is substantially different to approaches relying on a fixed number of sources listed in centralised catalogues. However, we believe that an approach based on decentralised linking is a feasible way for tackling the difficult challenge of accessing massive amounts of disparate data and systems, exacerbated in upcoming web of things environments (for example, smart energy grids, smart homes, smart vehicles or smart factories).

We address the individual challenges with the following contributions:

- C1: We provide methods and algorithms to access and query data in distributed environments [**P1**], and data that is created dynamically [**P2**].

- C2: We define several completeness classes for link traversal methods based on the correspondence in Linked Data between the identifier of a thing and the identifier of the data source with descriptions about the thing [**P3**].

- C3: We provide approaches to model temporal information, including an empirical analysis of how sources provide temporal information [**P4**].

- C4: We provide a language to specify link traversal and reasoning on Linked Data in concise, declarative means [**P5, P6**].

- C5: We provide means to also change/manipulate data and thus invoke actions within our framework **[P5]**.
- C6: We present scalable algorithms and methods to balance data access, query processing and reasoning on parallel hardware **[P6]**.

The presented contributions are pieces working towards a long-term goal of enabling autonomous behaviour of software agents in networked environments described next.

1.5 Overall Approach

Over the years, there have been many proposals for pieces of software that carry out tasks in large networked environments on behalf of its users. Kahn and Cerf, the "fathers of the internet" [13], already described such systems (dubbed "knowbots") in a project description in 1988 [31]. Etzioni and Weld proposed the notion of "softbots", involving artificial intelligence (AI) planning to carry out tasks for users on various internet resources in 1993 [18]. Wiederhold introduced the concept of "mediators" for accessing data from heterogeneous sources in 1996 [51]. Berners-Lee et al.'s 2001 article about the Semantic Web also included the notion of software agents [9].

We share the long-term goal of enabling integration and interoperation in networked systems. We argue that hyperlinks are the missing piece in achieving scale in terms of the number of sources and systems contributing to a common environment. Thus, our research focus is on methods for data access and manipulation in interlinked environments.

While it would be in principle desirable to have perfect data, the cost of achieving that level of quality can be prohibitive. Thus, the concept of pay-as-you-go data integration [21] has emerged: graph-structured data provides ways to view the data integration process as incremental, and integrate data as it is necessary to answer the information need. We want to take that idea one step further, and automatically take into account the data integration efforts of many contributors provided in a decentralised way on the web. As more and more data providers and authors add mappings to their data, the data consumers get an increasingly coherent and integrated view on the decentralised data. Rather than starting with data sources that are to be integrated, in our approach users start with a query that expresses an information need, and then try to find sources which provide data. The users then incrementally add mappings, until more and more results are returned for a query.

Existing approaches for autonomous access to sensors and effectors are often built on AI planning techniques, which require elaborate descriptions of sensors and effectors, as do approaches based on Semantic Web Service technologies [49]. While there are languages available for describing Web Services and Web APIs, one cannot assume large amounts of cleanly specified and described sources and effectors on the web[7]. As even the problem of providing integrated query access to interlinked graph-structured data is far from being solved and still an active area of research, we assume a simpler reactive model.

We base our work on an architecture that is data-driven (reactive) rather than goal-oriented; necessary descriptions are encoded into a rule-based specification, as opposed to assuming that the descriptions are attainable in a distributed way.

In our work, we assume that queries and link traversal specifications are evaluated at different points in time, given that the underlying data sources change [25]. In general, we calculate a fixpoint over the specifications. To be able to also incorporate state manipulation, consider two loops, taking ideas from cognitive architectures including Soar [45] and the widely deployed robotics control architecture from Brooks [12]. Figure 1 illustrates the high-level architecture with two main loops: the sense (polling) loop accesses the current state of the world, and the act loop changes the state of the world accordingly.

An instantiation of the sense cycle involves the following steps:

- Get the current snapshot of the state of the set of resources that represent the world.
- Materialise the data in the snapshot according to specified deduction rules.
- Follow links as specified in a rule-based program.
- Carry out queries over the materialised snapshot.

The sense cycle runs until a fixpoint has been reached, that is, the application of rules does not lead to new derived data or requests. Given that

[7]Changing the technologies used on the internet or the web takes years. Witness for example the slow adoption of IPv6, or the slow adoption of semantic annotations, from the specification of RDF Schema in 1999 [11] to a state where about 15 percent of web pages are annotated with data in 2013. `http://www.dataversity.net/schema-org-chat-googles-r-v-guha/`.

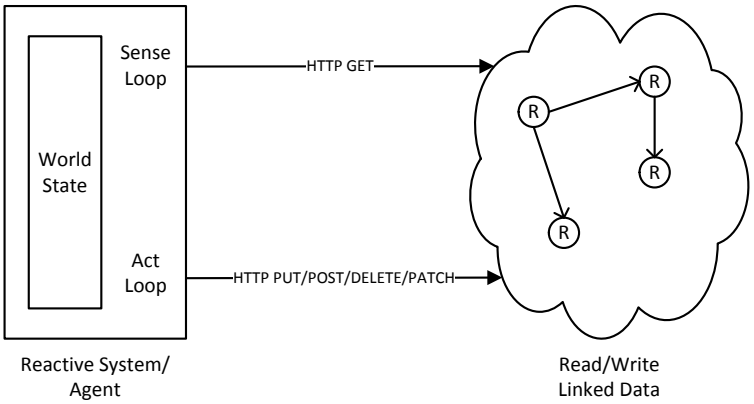


Figure 1: High-level architecture of a reactive link traversal system. The sense loop derives a snapshot of the current world state via HTTP GET operations on resources. Once the current snapshot has been derived, the system manipulates resource state via HTTP PUT, POST, DELETE and PATCH operations in the act loop.

data sources may change, users may choose to run that cycle at specified intervals to refresh query results, and so take into account the dynamics of data sources. In various prototypes, we operate on intervals as small as 33 ms (30 Hertz) [34, 35].

In case we want to also incorporate effectors (write access), we have to extend the interface to systems to allow for write access. Then, we can also extend the language to support unsafe HTTP requests[8]. Now, it is possible to trigger actions based on the materialised snapshot. With the appropriate extensions, we are able to carry out sequences of actions.

1.6 Outline

The remainder of the document is organised as follows.

Section 2 is mainly about query evaluation systems that leverage links to access and query Linked Data. We assume that the input from the user is just a query, and the method makes assumptions as to which sources to access. We mainly cover read-access to Linked Data, and explain how **[P1]**, **[P2]**, **[P3]** and **[P4]** fit into the overall picture. Section 2 addresses challenges C1-C3.

[8]PUT, POST, DELETE and PATCH.

Section 3 introduces our language for declarative specification of link traversal. Rather than just giving a query as input, the user specifies a query and a rule-based program that encodes how to derive new data and new requests [**P5**]. We also cover the ability to manipulate the state of resources. We further present algorithms required for scalable evaluation of those declarative link traversal specifications in [**P6**]. Section 3 addresses challenges C4-C6.

Section 4 concludes with a summary and an outlook.

2 Query Evaluation on Linked Data

Many systems that operate over RDF data require capabilities for query evaluation. SPARQL (a recursive acronym for SPARQL Protocol and RDF Query Language) is a query language for RDF specified by the W3C [44]. SPARQL allows for queries build on the notion of graph pattern matching (so-called Basic Graph Patterns, BGP), including conjunctions, disjunctions and optional patterns.

SPARQL includes support for dereferencing URIs to RDF graphs via HTTP, with the `FROM` and `FROM NAMED` clauses; a query processor may retrieve the representations of the graphs in the `FROM` and `FROM NAMED` clauses from the web before evaluating the query. However, all RDF graphs are retrieved before query evaluation starts; SPARQL thus does not provide facilities to follow links.

With the approach introduced by Hartig et al. [26], it is possible to answer certain SPARQL queries on Linked Data directly. Given a SPARQL query Q (with certain restrictions so that it is possible to start at sources with URIs specified in the BGP part of the query), the link traversal approach iteratively expands the set of sources by following the links specified in the BGP. While SPARQL queries posed over locally available indexed data take seconds to evaluate, evaluating queries over the Linked Data sources (while accessing the live sources at query processing time) may take minutes.

Some of the queries cannot be answered in the link traversal approach, because the query processor does not have any information about the available sources at all. At least one dereferencable URI in the query is required as starting point. During query evaluation, the method iteratively dereferences URIs bound to variables of the query. One of the drawbacks of the method is that it does not consider previously dereferenced graphs in subsequent runs. Another limitation is that the termination condition is specified via a heuristic. We address these limitations in the following two sections.

2.1 Querying Linked Data with Data Summaries

In [**P1**], we introduce a data structure to compactly store a summary of Linked Data sources. Such a summary can reduce the amount of data retrieved during query processing, and also reduce the amount of time spent traversing links, as previously discovered graphs are stored and immediately available during query processing. In the following, we sketch the structure of the data summary, how we build the summary, and how we use the summary in query evaluation.

The summary is based on a QTree data structure [30], which in principle is a combination of histograms with R-trees. As such, the QTree can store multidimensional data and capture attribute correlations. However, we cannot hold detailed information in the QTree due to storage requirements, and thus we approximate the information we hold using buckets. Thus, we group information together and thus might loose precision (we introduce false positives, but not false negatives). In an experiment carried out on a dataset of 3 million triples from about 16,000 sources with 516MB, the summary data structure requires around 22MB on disk, yielding a compression ratio of 96%. A triple in the QTree is represented as a point in a multidimensional space; a triple pattern is represented as a region.

The summary data structure can be based on data accessed via crawling, or on data accessed via link traversal. Combining both approaches elegantly solves the cold-start problem: a system could start with performing a plain link traversal approach, and successively expand the QTree summary with more relevant sources. As such, a system could expand its knowledge of the available interlinked data over time, and thus increase the coverage and efficiency of query answering gradually.

During query processing, the BGP of a query Q is split into its constituent triple patterns. Each triple pattern is then sent to the summary index, and the summary index returns the list of sources that may contain answers to the triple pattern. Please note that due to the approximate nature of the index, the returned list is a superset of the sources. A triple pattern (due to the use of variables) is represented as a region in multidimensional space.

For query processing, we access all the buckets that match a triple pattern, and which results in the list of sources. We can additionally rank the list of sources based on the estimated result cardinality, and therefore reduce the number of sources that have to be dereferenced (which is expensive). Let T be the total number of sources in the data, and E the number of estimated

sources. We calculate the benefit $1 - \frac{E}{T}$ that measures the number of sources that can be skipped compared to the naïve approach of simply accessing all known sources with zero benefit. We achieve benefits ranging from 20% to 80%, depending on the shape of the query. We also investigate the impact of ranking based on the estimated cardinality of sources stored in the QTree buckets. With the top-200 sources, we achieve recall of above 50% for four out of seven test query sets.

With the approach in **[P1]** we create an approximate description of the contents of arbitrary Linked Data sources.

2.2 Descriptions for Data Sources and Systems

While in general Linked Data sources provide arbitrary RDF graphs, there are many data sources that provide data in a regular structure. Consider large datasets converted from relational databases, or data exposed via Web APIs. Some data sources may not be published as a fully materialised knowledge base, because the underlying data may be constantly changing (e.g., sensor data), data is generated on the fly (e.g., distance between geographical points specified with arbitrary precision) or the data provider wants to restrict access to data (e.g., prices for flight tickets). Some of the sources have a simple query interface over data that follows a regular structure, and thus could be described. For essentially relational data sources that follow a fixed structure, we can devise descriptions that support linking and query processing.

In **[P2]**, we show how to describe a Linked Data interface over parameterised sources that are accessible via HTTP GET. We call the approach Linked Data Services (LIDS). We assume read-only resources that take name/value pairs as input, and return as output a regular RDF graph. We describe the input as a SPARQL BGP that returns query solution mappings, which can be viewed as name/value pairs. The shape of the output graph is also described as SPARQL BGP. Finally, the description includes the relation between input and output.

Rather than using the descriptions for backward-chaining, that is, going from the query Q to the sources D, in the paper we used a forward-chaining (data-driven) approach: we materialised all data that would match the service description. Such an approach has benefits in data collection scenarios, and can be used in a pipeline to enrich a dataset with data from the described services.

One of the lessons learned for the LIDS description language is that

while one can devise a description format for sources, actually creating and annotating sources following that interface and that specific description is a tedious task. Thus, in subsequent work, we decided to work on approaches that may use source descriptions, but do not require source descriptions to work.

Descriptions for data sources are difficult to do for graph-structured data in general [1]. However, since many sources actually follow a regular structure, recommendations such as R2RML (RDB to RDF Mapping Language) [16], and current work of the W3C RDF Data Shapes Working Group could lead to more descriptions available for RDF data sources, as the descriptions are not just provided as an extra functionality (and thus make effort for the data provider with only benefit for the data consumer), but are a core part of the data transformation itself on the side of the data provider.

With Semantic Web Services [49], the descriptions are typically based on IOPE (input-output-precondition-effect), to be able to model the structure of the messages sent (input, output), and also the change on the world state (precondition, effect). One major contribution of LIDS was to capture the relation between input and output both in the description as well as in the returned data, which IOPE descriptions did not provide. With that type of descriptions and data model (where there is a link in the input and the output data), it is possible to interpret the messages to and from the service as-is, that is, the data is self-describing. Current proposals for resource descriptions in the REST area are HAL [33], schema.org actions[9] and Hydra [38].

2.3 Completeness of Query Results

The web is vast, and we need a way to specify when a query has been answered completely. With a catalogue of descriptions of data sources (assuming the descriptions are complete), a query evaluation system can make precise statements about the completeness of the query results. Given the difficulty of getting descriptions of data sources on the web, and the lack of a central catalogue, we want to rely on hyperlinks to discover new sources and systems. In a hyperlinked environment without clearly defined boundaries, having a notion of completeness of query results is a challenging problem.

Possibilities to curb the amount of sources that are dereferenced include a limit on time and a limit on the number of dereferenced source [26]. With

[9] `https://schema.org/Action`

the approach of [**P1**], the system could be limited to dereference at most the sources in the summary index. In [**P3**] we propose several different completeness classes. Given a query, we have complete results with regard to a completeness class C if we have accessed all sources included in the dataset defined by C.

The broadest completeness class is web-complete, which is mainly of theoretic interest. With web-complete, we define queries as answered completely if all data sources on the web have been considered. As the web can be very large to infinite, in practice we will not be able to achieve web-complete answers to a query.

The next specific completeness class is seed-complete, which assumes that a query processor dereferences all sources that can be reached following triple paths of maximum length of the query, beginning from the seed URIs specified in the query.

For a more restricted completeness class, query-reachable, we exploit an idea of authoritativeness that has been investigated in [28] and [23]. We specify completeness based on the way how URIs are minted. There is a correspondence between the URIs in the data and the URIs of the data sources (either via HTTP redirects or via syntactic conventions on URIs). Exploiting that correspondence, we can use a user-supplied authority mapping to determine which sources to dereference to achieve complete results for a triple pattern, and then iteratively expand the completeness notion to entire BGPs. Thus, we can ensure that we have accessed all sources that correspond to the URIs in the data according to the specified authority mapping.

2.4 Temporal Information

The periodic evaluation of programs and queries leads to query results over time. When doing repeated evaluation of queries, we require accurate temporal information to track query results over time. In [**P4**], we carry out an empirical study on the different ways used in Linked Data sources of encoding temporal information. While some approaches assume the availability of temporal information in a specific way, we found little support for these approaches on the Linked Data web. In fact, even if temporal information was supplied, we sometimes found that the information was outdated or conflicting with other temporal information from the same source. Thus, we found that the availability of temporal information representing the history and temporal validity of statements and graphs is very limited. A lesson

learned was that for subsequent approaches involving dynamic data from the web, we cannot rely on temporal information supplied from the source. Given the rather messy state of temporal information on the web, in our subsequent work we do not assume the availability of accurate timestamps from the sources.

2.5 Related Work

Dataspace system aim at supporting the integration of distributed sources. The architecture of dataspace systems includes a catalogue in which all sources are listed, including schema information, statistics, change frequency, accuracy, completeness, capabilities of the query interface, ownership, and policies related to access and privacy [21]. In contrast, the web architecture does not rely on a catalogue, but relies on hyperlinks to link sources. Both descriptions of source and the sources' content are accessed via HTTP. Information such as change frequency are accessed via HTTP headers. With such an architecture, the sources are autonomous, and can change information about their rate of change without accessing a central catalogue. The sources describe themselves, and each source can provide mappings to other sources, expressed in RDFS or OWL.

Several link traversal systems exist that evaluate queries directly over sources accessible as Linked Data [26, 20, 36]. The work in **[P1]** provides mechanisms for optimising repeated evaluation of queries based on an in-memory index data structure. The work in **[P3]** provides a method for specifying the completeness of query results that goes beyond the heuristics in [26, 36].

3 Declarative Specifications for Link Traversal

We have seen how to answer a query Q over an implicitly specified dataset D. As we have hyperlinks, and D is large and diverse, we need finer-grained control over the semantics of the data and the way we want to expand links. Link traversal query approaches [26, 20, 36] **[P1]** require that the data in the sources exactly matches the query pattern, in terms of both schema URIs and the way how data is linked. One example for the exact match of a query pattern is the directionality of `owl:sameAs`: link traversal approaches assume that `owl:sameAs` is specified in the query as in the data, and do not take into account the properties of `owl:sameAs` (that of an equality relation). One example for the exact match regarding data linkage is that it is difficult

to specify that for a certain data source a link (`rdfs:seeAlso`, say) should be followed because the target source contains useful data. To be able to deal with the heterogeneity of the web with thousands of data authors, we require finer-grained control over the specification of both semantics of data items and link expansion.

The goal of our work **[P5]**, **[P6]** is to declaratively specify link traversal, and study parallel algorithms for evaluating these link traversal specifications. The basic framework for specifying link traversal are rules. We modify rules so that instead of asserting new knowledge in the consequent of a rule, we carry out HTTP requests if the condition in the antecedent holds. The data returned from requests is stored in memory. Thus, the system incrementally builds up the snapshot of the current state of the world. We can either evaluate a query Q over the state of the world (the dataset D), or trigger actions if a certain condition holds.

We define the syntax of declarative specifications for link traversal, and an operational semantics for such programs in **[P5]** and **[P6]**.

3.1 Rule-based Languages

We define logic-based languages (syntax and semantics) for integration and interoperation with Linked Data. The syntax of the rule language to specify programs is Notation3 [8], an extension to RDF with the ability for graph quoting and variables.

We can define different variants of the rule language with increasing expressivity:

- Level 0: the equivalent of positive Datalog (without function symbols) [14] on binary relations (or one ternary triple relation) with boolean built-in functions for specifying filter conditions. That language is guaranteed to terminate, as there are no ways to generate new symbols, and thus there is only a finite number of ways the symbols in a dataset can be combined.

- Level 1: Level 0, plus the ability to specify safe HTTP requests (GET) in rule heads; we also add built-in functions for simple arithmetic and string manipulation. Programs in this language will terminate under certain conditions.

- Level 2: Level 1, plus the ability to specify unsafe HTTP requests (PUT, POST, DELETE, PATCH) in rule heads. Programs in this language are not guaranteed to terminate.

In **[P5]**, we use state transition systems [37] to characterise the changes in Linked Data over time. Compared to the approach that operates on local state (e.g., the Game Description Language [39]), the state our programs operate on is the state of network-accessible remote resources.

3.2 Parallel Evaluation of Programs

In **[P6]**, we present algorithms for evaluating rule-based programs in a parallel fashion. Given that data volumes are large on the web, we have to devise scalable algorithms that exploit the recent developments in hardware: more main memory and multi-core systems. While hitherto users had to set up separate systems for data access, reasoning, and query processing into a pipeline, we have devised a method which integrates the three functionalities, and allows for efficient evaluation of the pipeline.

We experiment with two general approaches for the evaluation of recursive programs: one is a rounds-based method that can be run in batch, similar to the approach of Afrati and Ullman [2]. Another approach is a fully streaming model, related to the approach of Barish and Knoblock [4]. However, we include query optimisation techniques to reduce both the amount of work that has to be expended and the amount of storage space required.

In summary, we tackle the problem of parallelising the work based on dataflow graphs, and devise a method for detecting termination in dataflow graphs which contain cycles (in case of programs that contain recursive rules). We have shown that the integrated pipeline has competitive performance with existing reasoning systems, and has increased functionality for data access.

3.3 Related Work

Declarative rule languages can provide the foundation for both query and ontology languages that are used on the web [43, 29]. Our rule language covers both query and reasoning functionality. In addition, our rule language includes support for HTTP requests, and thus allows to specify behaviour next to declarative knowledge, similar to the Game Description Language [39]. Another related concept are Abstract State Machines (ASMs) [10], which can serve as basis for system design and analysis. In contrast, we use our specifications for generating behaviour involving the access and manipulation of the state of network-accessible resources. In other words, programs in our rule language provide the actual executable application logic.

Programming models such as MapReduce [17] can be used to specify data processing operations using higher-level functions map and reduce, and systems implementing that model are able to batch-process large amounts of data. In scenarios where multiple operations are specified, or in applications where batch-processing introduces too much latency, Spark [52] provides a processing model based on mini-batches and thus reduces end-to-end latency. However, these systems natively only support acyclic dataflows, and recursion would have to be built on top. Our systems supports the evaluation of cyclic dataflows based on queues that are used to break the cycle. There are various threading models with implications for termination detection. We plan to adapt our architecture and algorithms for termination detection to these distributed data processing systems as future work, to further scale up the amount of data that can be handled.

The system in [4] also uses a dataflow architecture to processing data for data integration. Rather than requiring wrappers to data sources and systems to view the underlying data in a relational model, we can assume access to many sources that provide a Linked Data interface. Following of hyperlinks in [4] is only supported in a very basic form, namely following "next" links in paged result pages, while in our approach the notion of request and hyperlink is accessible via rules.

Systems for link traversal [26, 20, 36] **[P1]** do not take the semantics of data sources (the mappings of schema and instance elements) into account during query processing. Other systems implementing reasoning [47, 41] typically operate over locally accessible single-source datasets; we assume a hyperlinked environment where data access and data processing are interleaved. Dataspace systems [21] rely on a centralised catalogue of sources; on the web, we assume hyperlinks between sources for resource discovery. Stream reasoning systems [40] and complex event processing systems [3] rely on a fixed number of sources that push data. On the web, polling is the prevalent communication mode. Further, in an environment based on resources and polling, we are able to discover new sources and new data (including new reasoning constructs) at runtime.

We have used the language and system in various demonstrators, from geospatial data integration [25], financial and statistical data integration [32] to specifying the interoperation in interactive virtual reality systems [35, 34].

4 Conclusion

The web is based on a scalable architecture with simple and minimal interfaces, with hyperlinks as one of the foundational concepts. While the internet and the web are fairly recent developments, logic-based knowledge representation, including logic programming, has a very long history. However, both fields have not been combined to the extent required to enable declarative access to the many data sources and systems accessible via web technologies. Neither hyperlinks nor logic-programming approaches are fully leveraged in current approaches for data integration and system interoperation based on a Linked Data interface. Often, RDF serves as an exchange format for database archives; multiple files are loaded into a SPARQL repository and then queried, without taking into account the semantics of data as specified using ontology languages such as RDFS or OWL. We argue for the use of rule-based programs to handle the semantics of ontology languages, provide means to specify link traversal, and support the inclusion of effectors or actions. To sum up, in this thesis we have made steps in bringing together logic-based knowledge representation with the traversal of hyperlinks and the invocation of actions, for a rudimentary specification of behaviour in complex information ecosystems.

We have described an environment with maximum flexibility and as little assumptions as possible. While we assume a wide-area network for communication, our methods can also be applied in local-area networks or in data centres. We assume very simple query capabilities at the sources, but given the unified communication protocol on the web, it is possible to dispatch queries to sources with more expressive query interfaces, such as those based on SPARQL or subsets thereof. We assume scenarios with many sources, but also only a few number of sources could be accessed, even a fixed set of sources. In these environments, however, different optimisation tradeoffs might apply, and less generic solutions might perform better.

Our work shares some of the goals with previous research on Semantic Web Services. One impediment to realising the vision outlined in the area of Semantic Web Services (such as OWL-S or WMSL [49]) was that the creation of elaborate domain models and descriptions of services never caught on with the wider web community. In contrast, there is an active community around Linked Data and another community around RESTful Web APIs that both work towards similar goals with a technology stack of reduced complexity, including a stronger focus on hyperlinks.

Data on the web is on the verge of becoming mainstream. Google's

Knowledge Graph[10] is an example for an application using data integrated from the web. Wikidata [50] is a system to manage factual data as basis for constructing infoboxes in Wikipedia. Schema.org[11] is an effort led by the big search engine companies to support the annotation of web pages with structured content. Schema.org provides a vocabulary for schema-level elements (of which there are only a few thousand), but not for instance-level elements (of which there may be several millions[12]). While these isolated efforts provide visibility and are a step in the right direction, the goal should be that all web users are able to access and integrate data from the web.

In the past, the internet and the web have integrated disparate networks such as CompuServe, AOL and BTX/Datex-J. Currently, there are many competing proposals for technologies and standards for the Internet of Things, Industrie 4.0, the Smart Grid and cyber-physical systems. Our expectation is that in time, internet and web technologies will serve as the basis for those new cyber-physical systems as well. In future application scenarios, it becomes increasingly important to have the ability to flexibly integrate new data sources and systems. Instead of a web browser as universal access to information and functionality, virtual assistants, such as Amazon's Echo[13], Apple's Siri (originally IRIS [15]), Facebook's M[14], Google's Now[15], Microsoft's Cortana[16] or Samsung's S-Voice[17], can provide a universal user interface to data and functionality from multiple sites. However, if we do not want a single company dominating how we access the web (of documents, data, things), the underlying technologies have to remain open, and the barrier to entry should be low. Everybody should have a chance to leverage the vast amounts of information attainable via the web. We believe that the approaches and methods we have presented fit well with these current developments, and will make it easier for people to access and manipulate the wealth of information and services online.

[10] `https://google.com/intl/bn/insidesearch/features/search/knowledge.html`

[11] `http://schema.org/`

[12] While the ultimate goal of schema.org might be to support identifiers for instance-level elements (entities), the current mantra is "strings for things", to avoid the need for content providers to find identifiers for entities. However, the lack of shared identifiers for entities requires a disambiguation and linking step before such data can be used as a Knowledge Graph.

[13] `http://www.amazon.com/oc/echo/`

[14] `http://www.wired.com/2015/08/facebook-launches-m-new-kind-virtual-assistant/`

[15] `https://www.google.com/landing/now/`

[16] `http://www.microsoft.com/en/mobile/experiences/cortana/`

[17] `http://www.samsung.com/global/galaxys3/svoice.html`

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, October 1999.

[2] F. N. Afrati and J. D. Ullman. Transitive closure and recursive datalog implemented on clusters. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 132–143, 2012.

[3] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012.

[4] G. Barish and C. A. Knoblock. An expressive language and efficient execution system for software agents. *Journal of Artificial Intelligence Research*, pages 625–666, 2005.

[5] R. Bennett, C. Hengel-Dittrich, E. T. ONeill, and B. B. Tillett. VIAF (virtual international authority file): Linking Die Deutsche Bibliothek and Library of Congress name authority files. In *World Library and Information Congress: 72nd IFLA General Conference and Council*, 2006.

[6] T. Berners-Lee. Linked Data, 2006. `http://www.w3.org/DesignIssues/LinkedData`.

[7] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, Aug. 1994.

[8] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3logic: A logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8(3):249–269, May 2008.

[9] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[10] E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

[11] D. Brickley and R. V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-schema/`.

[12] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[13] V. G. Cerf. The day the Internet age began. *Nature*, 461:1202–1203, 2009.

[14] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1:146–166, March 1989.

[15] A. Cheyer, J. Park, and R. Giuli. IRIS: Integrate. Relate. Infer. Share. In *Proceedings of the 1st Workshop on The Semantic Desktop*, Nov. 2005.

[16] S. Das, S. Sundara, and R. Cyganiak, editors. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation, September 2012. `http://www.w3.org/TR/rdf-schema/`.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008.

[18] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, July 1994.

[19] L. Feigenbaum, I. Herman, T. Hongsermeier, E. Neumann, and S. Stephens. The semantic web in action. *Scientific American*, 297(6):90–97, 2007.

[20] V. Fionda, C. Gutierrez, and G. Pirró. Semantic navigation on the web of data: Specification of routes, web fragments and actions. In *Proceedings of the 21st International Conference on WWW*, pages 281–290, 2012.

[21] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *SIGMOD Record*, 34(4):27–33, Dec. 2005.

[22] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, Mar. 2009.

[23] A. Harth, S. Kinsella, and S. Decker. Using naming authority to rank data and ontologies for web search. In *Proceedings of the 8th International Semantic Web Conference (ISWC)*, 2009.

[24] A. Harth, C. A. Knoblock, K. Sattler, and R. Studer. Interoperation in complex information ecosystems (Dagstuhl seminar 13252). *Dagstuhl Reports*, 3(6):83–134, 2013.

[25] A. Harth, C. A. Knoblock, S. Stadtmüller, R. Studer, and P. A. Szekely. On-the-fly integration of static and dynamic sources. In *Proceedings of the 4th International Workshop on Consuming Linked Data (COLD)*, 2013.

[26] O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference (ISWC)*, pages 293–309, 2009.

[27] C. Hewitt. The challenge of open systems: Current logic programming methods may be insufficient for developing the intelligent systems of the future. *BYTE*, 10(4):223–242, Apr. 1985.

[28] A. Hogan, A. Harth, and A. Polleres. Scalable authoritative OWL reasoning for the web. *International Journal on Semantic Web Information Systems*, 5(2):49–90, 2009.

[29] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers? In *Principles and practice of semantic web reasoning*, pages 37–41. Springer, 2005.

[30] K. Hose, M. Karnstedt, A. Koch, K.-U. Sattler, and D. Zinn. Processing rank-aware queries in p2p systems. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 171–178. Springer, 2007.

[31] R. E. Kahn and V. G. Cerf. An open architecture for a digital library system and a plan for its development. Technical report, Corporation for National Research Initiatives, 1988.

[32] B. Kämpgen, T. Weller, S. O'Riain, C. Weber, and A. Harth. Accepting the XBRL challenge with linked data for financial data integration. In *Proceedings of the 11th International European Semantic Web Conference (ESWC)*, pages 595–610, 2014.

[33] M. Kelly. HAL - hypertext application language: A lean hypermedia type. Technical report, Stateless.co, 2011, 2013.

[34] F. L. Keppmann, T. Käfer, S. Stadtmüller, R. Schubotz, and A. Harth. High performance linked data processing for virtual reality environments. In *Proceedings of the Posters & Demonstrations Track of the 13th International Semantic Web Conference (ISWC)*, pages 193–196, 2014.

[35] F. L. Keppmann, T. Käfer, S. Stadtmüller, R. Schubotz, and A. Harth. Integrating highly dynamic RESTful linked data APIs in a virtual reality environment. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 347–348, 2014.

[36] G. Ladwig and T. Tran. Linked data query processing strategies. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, pages 453–469, 2010.

[37] L. Lamport. Computation and state machines. Technical report, Microsoft Research, 2008.

[38] M. Lanthaler. Hydra core vocabulary: A vocabulary for hypermedia-driven web APIs. Technical report, Google, 2015.

[39] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General game playing: Game description language specification. Technical report, Stanford Logic Group, 2008.

[40] A. Margara, J. Urbani, F. van Harmelen, and H. Bal. Streaming the web: Reasoning over dynamic data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 25(0), 2014.

[41] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *Proc. of the 28th AAAI Conference on Artificial Intelligence*, pages 129–137, 2014.

[42] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big" web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, pages 805–814, 2008.

[43] A. Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web*, pages 787–796, 2007.

[44] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF, Jan. 2008. W3C Recommendation, `http://www.w3.org/TR/rdf-sparql-query/`.

[45] P. S. Rosenbloom, J. E. Laird, and A. Newell, editors. *The Soar Papers (Vol. 1): Research on Integrated Intelligence*. MIT Press, 1993.

[46] G. Schreiber and Y. Raimond, editors. *RDF 1.1 Primer*. W3C Working Group Note, 24 June 2014. `http://www.w3.org/TR/rdf11-primer/`.

[47] M. Sintek and S. Decker. TRIPLE - A query, inference, and transformation language for the semantic web. In *Proceedings of 1st International Semantic Web Conference (ISWC)*, pages 364–378, 2002.

[48] S. Speicher, J. Arwe, and A. Malhotra, editors. *Linked Data Platform 1.0*. W3C Proposed Recommendation, 16 December 2014. `http://www.w3.org/TR/ldp/`.

[49] R. Studer, S. Grimm, and A. Abecker, editors. *Semantic Web Services - Concepts, Technologies, and Applications*. Springer, 2007.

[50] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, Sept. 2014.

[51] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, Mar. 1992.

[52] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2012.

List of Publications

[P1] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data Summaries for On-demand Queries over Linked Data. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 411–420. 2010.

[P2] Sebastian Speiser and Andreas Harth. Integrating Linked Data and Services with Linked Data Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, pages 170–184. 2011.

[P3] Andreas Harth and Sebastian Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*. 2012.

[P4] Anisa Rula, Matteo Palmonari, Andreas Harth, Steffen Stadtmüller, and Andrea Maurino. On the Diversity and Availability of Temporal Information in Linked Open Data. In *Proceedings of the 11th International Semantic Web Conference (ISWC)*, pages 492–507. 2012.

[P5] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 1225–1236. 2013.

[P6] Andreas Harth and Steffen Stadtmüller. Parallel Processing of Rule-based Programs on Linked Data. 2015. Under review.

[P1] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data Summaries for On-demand Queries over Linked Data. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 411–420. 2010.

Data Summaries for On-Demand Queries over Linked Data*

Andreas Harth[#], Katja Hose[*], Marcel Karnstedt[‡], Axel Polleres[‡], Kai-Uwe Sattler[†], Jürgen Umbrich[‡]

[#]AIFB, Karlsruhe Institute of Technology, Germany
[*]Max-Planck Institute for Informatics, Saarbrücken, Germany
[‡]Digital Enterprise Research Institute, National University of Ireland, Galway
[†]Ilmenau University of Technology, Ilmenau, Germany
[#]harth@kit.edu, [*]hose@mpi-inf.mpg.de, [‡]*firstname.lastname*@deri.org, [†]kus@tu-ilmenau.de

ABSTRACT

Typical approaches for querying structured Web Data collect (crawl) and pre-process (index) large amounts of data in a central data repository before allowing for query answering. However, this time-consuming pre-processing phase however leverages the benefits of Linked Data – where structured data is accessible live and up-to-date at distributed Web resources that may change constantly – only to a limited degree, as query results can never be current. An ideal query answering system for Linked Data should return current answers in a reasonable amount of time, even on corpora as large as the Web. Query processors evaluating queries directly on the live sources require knowledge of the contents of data sources. In this paper, we develop and evaluate an approximate index structure summarising graph-structured content of sources adhering to Linked Data principles, provide an algorithm for answering conjunctive queries over Linked Data on the Web exploiting the source summary, and evaluate the system using synthetically generated queries. The experimental results show that our lightweight index structure enables complete and up-to-date query results over Linked Data, while keeping the overhead for querying low and providing a satisfying source ranking at no additional cost.

Categories and Subject Descriptors:
E.1[Data Structures]: Distributed Data Structures; H.2.4[Database Management]: Systems—Distributed Databases, Query Processing;

General Terms: Algorithms, Design, Performance

Keywords: Index Structures, Linked Data, RDF Querying

1. INTRODUCTION

The recent developments around Linked Data promise to lead to the exposure of large amounts of data on the Semantic Web amenable to automated processing in software programs [1]. Linked Data sources use RDF (Resource Description Format) in various serialisation syntaxes for encoding graph-structured data. The Linked Data effort is part of a trend towards highly distributed systems, with thousands or potentially millions of independent sources providing small amounts of structured data. Using the available data in data integration and decision-making scenarios requires query processing over the combined data.

For evaluating queries in such environments we can distinguish two directions:

- data warehousing or *materialisation-based approaches (MAT)*, which collect the data from all known sources in advance, preprocess the combined data, and store the results in a central database; queries are evaluated using the local database.
- *distributed query processing approaches (DQP)*, which parse, normalise and split the query into subqueries, determine the sources containing results for subqueries, and evaluate the subqueries against the sources directly.

Unfortunately, applying DQP directly is not a viable solution for Linked Data sets: firstly, in most cases the data in the different sources cannot be described by simple expressions because they may vary in the schema or do not even have common values. Secondly, queries cannot be "dispatched", unless query processing capabilities exist at the source sites. Preliminary results for distributed query processing over distributed RDF sources [25] assume, similar to resp. approaches from the traditional database works, relatively few query endpoints with probably huge amounts of data, rather than many small Web resources accessible via simple HTTP GET only.

The aim of the present paper is to narrow the gap between these two extreme approaches and find a reasonable middle-ground for processing queries over Linked Data sources directly. Although currently only a few data sources offer full query processing capabilities (e.g., by implementing SPARQL [4, 24], a query language and protocol for RDF), we still can eschew the cost of maintaining a full index of the data at a central location. On the current Web, all we can assume is that the sources implement a single operation *GET* which returns the content of the source in RDF. Thus, instead of full federation we propose an approximate multidimensional indexing structure (a QTree [14]) to store descriptions of the content of data sources. The QTree forms the basis for sophisticated query optimisation and helps the query processor decide on which sources to route a query or a subquery. We assume – as typical for Linked Data – a large number of sources, which, in contrast to classic data integration scenarios, are of small size in the range of a few kilobytes to megabytes.

*This material is in parts supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and 08/SRC/I1407 (Clique) and the EU under projects NeOn (IST-2006-027595) and ACTIVE (IST-2007-215040). We thank Aidan Hogan for comments.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

Approximate data summaries such as QTrees can be populated by crawling techniques similar to those employed by centralised systems, with the advantage of a significantly smaller index, which can be kept in memory, and live query results, by processing the actual query only over those sources which likely contain relevant information. Also, such a QTree index can be dynamically extended, by adding either user-submitted sources or sources discovered during query processing.

The strategy we propose is a reasonable compromise under the assumption that the overall data distribution does not change dramatically over time: that is, the distribution characteristics are relatively stable, which holds for a wide range of Linked Data sources (e.g., DBpedia[1], DBLP[2], or machine-readable personal homepages). Under this assumption we can employ an approach which stores a data summary reflecting these immutable characteristics in lieu of a full local data index.

Our approach works as follows:

• prime an approximate index structure (a QTree) with a seed data set (various mechanisms for creating and maintaining the index are covered in Section 4);

• use the QTree to determine which sources contribute partial results for a conjunctive SPARQL query Q;

• fetch the content of the sources (optionally using only the top-k sources according to cardinality estimates stored in the QTree) into memory;

• perform join processing locally, given that remote sources do not provide functionality for computing joins.

The main problems of processing such queries hence become i) finding the right sources to contain possible answers that can contribute to the overall query and ii) efficient parallel fetching of content from these sources.

We conclude this section by introducing example data and queries used throughout the paper. Section 2 discusses alternative methods for answering queries over Linked Data. In Section 3, we present an approach to select sources from a QTree.Section 4 describes approaches to construct and maintain these data summaries followed by a discussion of the results of an evaluation in Section 5. In Section 6, we align our system with existing work and conclude with an outlook to future work in Section 7.

Example. As an example consider a scenario in which sources publish interlinked data about people, the relations between them and their publications. Such data is indeed available as Linked Data in RDF on the Web in the form of hand-crafted files in the Friend-of-a-Friend (FOAF) vocabulary [2] and automatic exports of publication databases such as DBLP.

For instance, consider the Linked Data sources depicted in Figure 1. RDF graphs comprise of (`subject predicate object`) triples that denote labelled edges between the subject and the object. The figure shows five RDF graphs covering data about Andreas and Axel: personal homepages encoded in FOAF, data covering personal information and one of their joint publications at DBLP. We assume that `namespace:localname` pairs expand to full URIs, e.g., `dblp:Axel_Polleres` expands to `http://dblp.l3s.de/d2r/resource/authors/Axel_Polleres`.

[1] `http://dbpedia.org/`
[2] `http://dblp.l3s.de/d2r/`

Conjunctive SPARQL queries[3] consist of so-called *basic graph patterns* (BGPs), i.e., sets of triple patterns containing variables. For instance, the following query asks for names of Andreas' friends:

```
SELECT ?n WHERE {
  andreas:foaf#ah foaf:knows ?f. ?f foaf:name ?n. }
```
(1)

The next query asks for authors of article `dblppub:HoganHP08` who mutually know each other:

```
SELECT ?x1 ?x2 WHERE {
  dblppub:HoganHP08 dc:creator ?a1, ?a2.
  ?x1 owl:sameAs ?a1. ?x2 owl:sameAs ?a2.
  ?x1 foaf:knows ?x2. ?x2 foaf:knows ?x1. }
```
(2)

2. QUERYING LINKED DATA

Linked Data [1] is RDF published on the Web according to the following principles: 1) use URIs as names for things 2) use (dereferenceable) HTTP URIs, 3) provide useful content at these URIs encoded in RDF, and 4) include links to other URIs for discovery. In the same way the current Web is formed by HTML documents and hyperlinks between documents, the Linked Data Web is constructed by using HTTP URIs (principle 1 and 2). Principle 3 – providing meaningful content for dereferenced URIs (that is, RDF triples describing the URI, typically in the subject position) – allows for a new way of performing lookups on the data during query runtime. The principle provides a correspondence (in URI syntax or via redirects in the HTTP protocol) between a URI of a resource and the data source. For example, the resource URI `http://dblp.l3s.de/d2r/resource/authors/Axel_Polleres` redirects to the source URI `http://dblp.l3s.de/d2r/page/authors/Axel_Polleres`. Finally, reusing URIs across sources (principle 4) makes sure that data covering the same entity can be collated from multiple sources.

Most current approaches enabling query processing over RDF data operate very much along the lines of relational data warehouses or search engines; Semantic Web search engines [3, 6, 13, 21] crawl large amounts of RDF documents for materialisation and indexing in a centralised data store.

The centralised approaches using materialisation (MAT) provide excellent query response times due to the large amount of preprocessing carried out during the load and indexing steps, but suffers from a number of drawbacks. First, the aggregated data is never current as the process of collecting and indexing vast amounts of data is time-consuming. Second, from the viewpoint of a single requester with a particular query, there is a large amount of unnecessary data gathering, processing, and storage involved since a large portion of the data might not be used for answering that particular query. Furthermore, due to the replicated data storage, the data providers have to give up their sole sovereignty on their data (e.g., they cannot restrict or log access any more since queries are answered against a copy of the data).

On the other end of the spectrum, there are approaches that assume processing power attainable at the sources themselves (DQP), which could be leveraged in parallel for query processing. Such distributed or federated approaches [11] offer several advantages: the system is more dynamic with up-to-date data and new sources can be added

[3] We focus on the core case of conjunctive queries and do not consider more complex features such as unions, outer joins, or filters available in SPARQL, which could be layered on top of conjunctive query functionality.

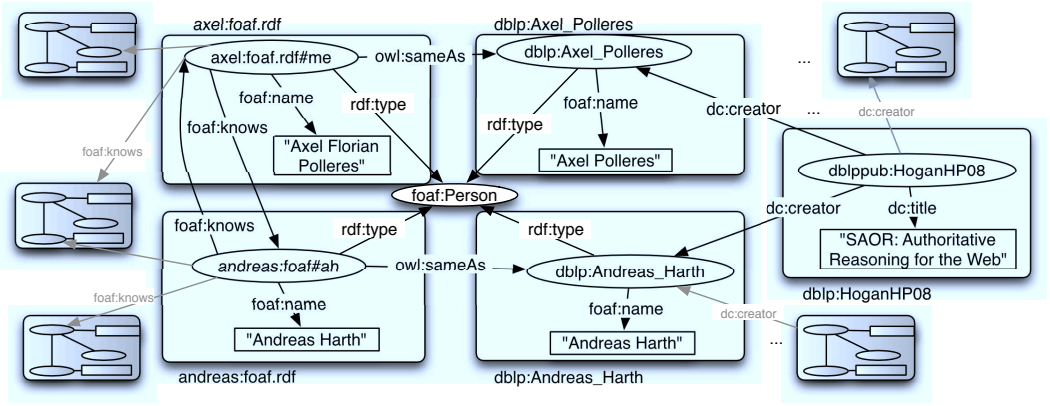


Figure 1: Linked Data in RDF about persons and their publications

easily without time lag for indexing and integrating the data, and the systems require less storage and processing resources at the query issuing site. The potential drawback, however, is that DQP systems cannot give strict guarantees about query performance since the integration system relies on a large number of potentially unreliable sources. DQP is a well-known database problem [17]. Typically, DQP involves the following steps for transforming a high-level query into an efficient query execution plan: parsing, normalising by application of equivalence rules, unnesting and simplification of the query, data localisation, optimisation (i.e., replacing the logical query operators by specific algorithms and access methods as well as by determining the order of execution both at a global and local level), and finally execution. Besides optimisation, data localisation is an important step that affects the efficiency of the execution. The goal of data localisation – also known as *source selection* – is to identify the source sites that possibly provide results for the given query or, in other words, to eliminate sites from the query plan that do not contribute to the result. In classic distributed databases this step is supported by (query or view) expressions describing the fragmentation of a global table.

Possible approaches to evaluate queries over such Web resources and particularly addressing the problem of source selection are:

• **Direct Lookups (DL)** The direct lookup approach is implemented in [10] where one tries to leverage the correspondence between source addresses and identifiers contained in the sources to answer queries. The query processor performs lookups on the sources that contain identifiers mentioned in the query or are retrieved in subsequent steps. To answer query (1) of Section 1, one could fetch content from `andreas:foaf#ah`, dereference `foaf:knows` links, and gather new information where hopefully the respective names of friends are found. The sources in the DBLP realm are irrelevant for answering this query. However, the strategy fails to find the solutions for query (2) since the necessary `owl:sameAs` links come from outside the linked closure of the graph `dblppub:HoganHP08`. Apart from possible incompleteness issues, the approach also has limitations in the sense that only limited parallelisation is possible: the query processor starts with one source and iteratively performs more lookups on sources determined by intermediate results rather than looking up the entire list of relevant sources in a single pass. On the positive side, if one can live with partial results this approach has no need for maintaining indexes since only the correspondence between source and contained identifiers is used.

• **Schema-Level Indexes (SLI)** A second approach, mainly based on distributed query processing, relies on schema-based indexes [7, 26]. The query processor keeps an index structure with properties (i.e., predicates) and/or classes (i.e., objects of `rdf:type` triples) that occur at certain sources, and uses that structure to guide query processing. Using such schema-based indexes the incompleteness problem of direct lookups is alleviated while only using lightweight index structures. The drawback is that instance-level descriptions are missing: i.e., i) only queries which contain schema-level elements can be answered, and ii) on very commonly used properties (e.g., `foaf:knows`, `foaf:name`), this index selects a (possibly too) large portion of all possible sources.

• **Data Summaries (DS)** A third approach, and the one we are advocating in this paper, uses a combined description of instance- and schema-level elements to summarise the content of data sources. We cannot keep every data item in this index, so we use a summarising index – a data summary – which represents an approximation of the whole data set. The DS approach uses more resources than the schema-level indexes, however, adds the ability to cover also query patterns including instance-level queries. Since the DS return sources which possibly contain answers to a query directly (i.e., taking joins into account), this approach may be viewed as subsuming both direct lookups and schema-level indexes. Further, a data summary index can be updated incrementally as the query processor obtains new or updated information about sources.

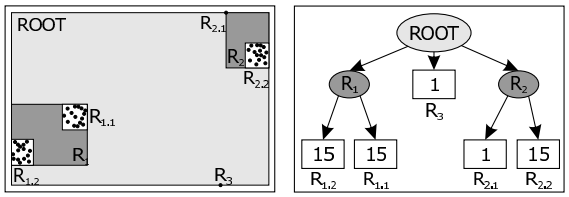
3. SOURCE SELECTION USING DATA SUMMARIES

Our main idea for identifying relevant sources is to index RDF triples provided by the sources by first transforming them into a numerical data space (applying hash functions) and then indexing the resulting data items with a data summary. In our work, we use an index structure called QTree – originally developed for top-k query processing [14, 15, 27] –

as our data summary. In the following, we describe the basic principles of this structure as well as its usage for source selection.

3.1 Source Indexing using the QTree

In principle, the QTree (Figure 2) is a combination of histograms and R-trees [8] inheriting the benefits of both data structures: indexing multidimensional data, capturing attribute correlations, dealing with sparse data, offering efficient look-ups, and supporting incremental construction and maintenance. Like the R-tree, a QTree is a tree structure consisting of nodes defined by minimal bounding boxes (MBBs). These MBBs describe multidimensional regions in the data space and MBBs of all nodes always cover all MBBs of their children and the subtrees rooted by them. Because R-trees are used to manage data items, leaf nodes in R-trees contain the data items that are contained in their MBBs. However, for our purposes we cannot hold detailed information about all data items. Rather, we have to reduce memory consumption by approximating this information.



(a) Data and Regions (b) Hierarchy

Figure 2: Two-dimensional QTree example

Thus, to limit memory and disk consumption, we replace subtrees with special nodes called buckets. Buckets correspond to histogram buckets or bins and are always leaf nodes in the QTree – and leaf nodes are always buckets. Data items are represented by the buckets in an approximated version. Since the construction of the QTree aims at grouping data items with similar hash values into the same bucket, we can use the MBBs as a good basis for approximation. As mentioned above, in our case data items are points in the multidimensional space whose coordinates are obtained by applying hash functions to the individual components (S, P, O) of RDF triples. These components correspond to dimensions in a three-dimensional QTree.

Only buckets contain statistical information about the data items contained in their MBBs. In principle, a bucket might hold any kind of statistics, but for the purpose of this work we consider buckets capturing the count of data items contained in their MBBs. Each bucket stores the number of triples whose values (`subject predicate object`) are mapped onto coordinates that are part of the bucket's MBB – the MBB being defined by $[S.low, S.hi], [P.low, P.hi], [O.low, O.hi]$.

The total number of buckets, as well as the size of a QTree, can be controlled by two parameters: i) b_{max} denoting the maximum number of buckets in the QTree and thus limiting memory consumption, ii) f_{max} describing the maximum fanout (i.e., the number of child nodes) for each non-leaf node. Note that the size of a QTree only depends on these two parameters and is independent from the number of represented data items.

Details on constructing and maintaining a QTree are beyond the scope of this paper. Thus, in the following we only sketch the basic idea and refer the interested reader to [15]. The QTree is constructed incrementally by inserting one data item after another. For each data item p, we first check whether it can be added to an existing bucket that encloses p's coordinates. In this case, the bucket statistics are updated by incrementing the number of contained data items. Otherwise, we traverse the QTree beginning at the root node in each level looking for a node whose MBB completely encloses p. Once we have arrived at a node whose children's MBBs do not contain p, we create a new bucket for p and insert it as a new child node.

In order to enforce the two constraints b_{max} and f_{max}, we have to merge buckets and child nodes if the number of buckets in the QTree or the fanout of inner nodes violates the constraints. For this purpose, we use a penalty function that represents the approximation error caused by merging two buckets and merge the pair of sibling buckets that minimises the penalty. The expensive check of all pairs is avoided by maintaining a priority queue.

To capture details on which RDF triples are provided by which source, we store not only the number of data items per bucket but also the URIs of sources whose triples are represented by the bucket. Basically, there are two possible approaches: i) we can simply keep a list $\mathcal{S}_B$ of source URIs and a bucket cardinality c_B, or ii) we maintain the number of triples c_B^s in each bucket B per source $s \in \mathcal{S}_B$, i.e., each bucket B contains a list of s, c_B^s pairs. For ease of explanation, in the following we stick to the first approach. In Section 3.2.2, we pick up the second approach, as it allows for a more sophisticated estimation of the number of results a source contributes to.

3.2 Source Selection

Let us now discuss how to use the information provided by the QTree to decide on the relevance of sources for answering a particular query.

3.2.1 Triple Pattern Source Selection

As joins are expressed by conjunctions of multiple triple patterns and associated variables, a prerequisite for join source selection is the identification of relevant sources for a given triple pattern.

To determine relevant sources we first need to identify the region in data space that contains all possible triples matching the pattern. Therefore, we need to convert a triple pattern into a set of coordinates in data space, using the same hash functions that we used for index creation, to obtain coordinates for a given RDF triple. However, in contrast to obtaining hash values for RDF triples provided by the sources, triple patterns of queries might contain variables. Because of these variables, in general we have to work with regions instead of points. Thus, for each literal, blank node or URI in a given triple pattern, we apply the hash functions and use the obtained hash values as minimum and maximum coordinates to define the queried region. For each variable, we set the minimum and maximum coordinates to the minimum/maximum possible hash values in the respective dimensions.

After having determined the queried region R, we only need to find all buckets in the QTree that overlap R. As the QTree – similar to the R-tree – has a hierarchical structure, the lookup procedure follows similar rules: starting at the

root node we need to traverse child nodes if their MBBs overlap R until we arrive at the buckets on leaf level.

After having identified all buckets with overlapping MBBs, we determine the percentage of overlap with R. Let size(R) denote the size of a region R, c_B the number of data items (cardinality) represented by bucket B and O the overlapping region of B and R. Then, the cardinality of O is calculated as $c_B \cdot \frac{\text{size}(O)}{\text{size}(B)}$. Based on the overlap, the bucket's source URIs, and the cardinality (i.e., the number of represented RDF triples) we can determine the set of relevant sources and the expected number of RDF triples per source – assuming that triples are uniformly distributed within each bucket. Thus, the output of the source selection algorithm is a set of buckets, each annotated with information about the overlap with the queried region, source URIs, and the associated cardinality.

3.2.2 Join Source Selection

In order to determine which sources provide relevant data for a join query, we first need to separately consider the triple patterns (BGPs) that a join query consists of. In principle, we could return the union of all sources relevant for the individual BGPs (Section 3.2.1) as the result of the join source selection. However, it is likely that there are no join partners for data provided by some of the sources, although they match one BGP. Thus, we consider the overlaps between the sets of obtained relevant buckets for the BGPs with respect to the defined join dimensions and determine the expected result cardinality of the join.

The crucial question is how we can discard any of the sources relevant for single BGPs, i.e., identify them as irrelevant for the join. Unfortunately, if a bucket is overlapped, we cannot omit any of the contributing sources, because we have no information on which sources contribute to which part of the bucket. To not miss any relevant sources, we can only assume all sources from the original bucket to be relevant. Sources can only be discarded if the entire bucket they belong to is discarded, such as the smaller bucket for the second BGP in Figure 3.

The result of a join evaluation over two BGPs is a set of three-dimensional buckets. Joining a third BGP requires a differentiation between the original dimensions, because the third BGP can be joined with any of them. For instance, after a subject-subject join we have to handle two different object dimensions; a join between two three-dimensional overlapping buckets results in one six-dimensional bucket with an MBB that is equivalent to the overlap. In general, a join between n BGPs results in a $(3 \cdot n)$-dimensional join space.

Figure 3 illustrates the first step of join source selection on example query (2) of the introduction, assuming that the first join is processed over the triples for subject `?x1`. For illustration purposes, we only show subject and object dimensions, as the predicate is fixed in both BGPs (i.e., the figures correspond to a slice of the three-dimensional space). Figure 3 illustrates a bucket that corresponds to the result of the source selection algorithm for the first BGP and shows two buckets corresponding to the second BGP. Both overlapping buckets are constrained by their overlap in the join dimension, which is the subject dimension. Other dimensions are not constrained. Thus, the shaded parts of both buckets represent the result buckets of the join.

Figure 4 illustrates the next join for example query (2), assuming that it is processed on `?x2` (object-subject join

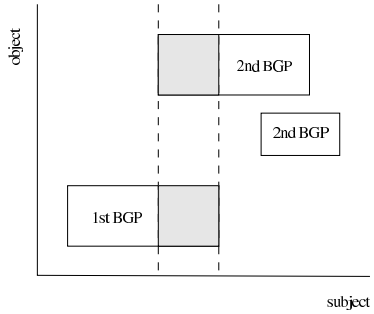


Figure 3: QTree join between first and second BGP

between 2nd and 3rd BGP). Again, for illustration purposes, we omit the predicate dimensions and show equal dimensions on the same axis (slices of the six-dimensional space reduced to the three shown dimensions).

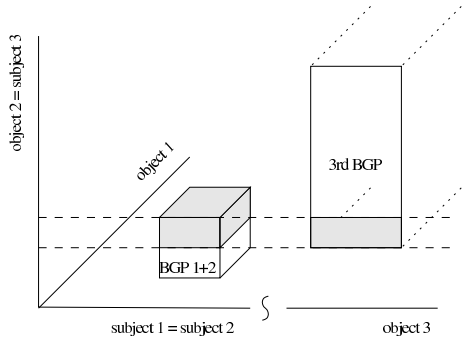


Figure 4: QTree join with third BGP

Algorithm 1 sketches the whole algorithm for join source selection. In general, source selection will result in multiple buckets for each BGP. The overlap has to be determined for the cross-product of all input buckets (lines 6 and 7). We determine the buckets for each BGP separately and join them afterwards (line 7), which allows us to use existing methods for determining the overlap between the resulting buckets.

The loop in line 5 shows that we process all joins sequentially, storing the results in variables $join_i$. We insert the result buckets of join i into a new $(3 \cdot (i+1))$-dimensional join space $join_i$. Note that, after the first join, two of the six dimensions are equal. Handling them separately is just for ease of understanding and implementation. The $\oplus$ operator in line 12 symbolises the operation of combining two buckets while increasing the number of dimensions accordingly: the three dimensions from O_R are added to the $3 \cdot i$ dimensions of O_L, together forming the $3 \cdot (i+1)$ dimensions of the result bucket. The new cardinality $c_{O_R \oplus O_L}$ (line 11) of the resulting bucket is determined using the percentage of overlap for both buckets (cf. Section 3.2.1 and line 3) and assuming uniform distribution in both buckets. The set of relevant sources $\mathcal{S}_{O_R \oplus O_L}$ is a union over the sets from both buckets. Finally, $join_i$ serves as input for the next join (line 6).

3.3 Source Ranking

As source selection is approximate, the set of relevant sources will usually be overestimated, i.e., contain false positives. Please note that false negatives are impossible as we consider all QTree buckets matching any part of the query.

Input: Query q, QTree QT
Output: list of relevant sources

1 **forall** buckets $B \in QT$.getBuckets(q.BGP[0]) **do**
2 $O = B$.overlap(q.BGP[0]);
3 $join_0$.insert($O, c_B \cdot \frac{\text{size}(O)}{\text{size}(B)}$, $\mathcal{S}_B$);
end
5 **for** $i = 1$ to $|q.\text{BGP}| - 1$ **do**
6 **forall** buckets $L \in join_{i-1}$ **do**
7 **forall** buckets $R \in QT$.getBuckets(q.BGP[i]) **do**
8 $d_L = q$.joindim[$i-1$]; $d_R = q$.joindim[i];
9 **if** $\exists O_L = L[d_L]$.overlap($R[d_R]$) **then**
10 $O_R = R[d_R]$.overlap($L[d_L]$);
11 $c_{O_R \oplus O_L} = \frac{c_L \cdot \frac{\text{size}(O_L)}{\text{size}(L)} \cdot c_R \cdot \frac{\text{size}(O_R)}{\text{size}(R)}}{\max(L[d_L].hi - L[d_L].low, R[d_R].hi - R[d_R].low)}$;
12 $join_i$.insert($O_L \oplus O_R, c_{O_R \oplus O_L}, \mathcal{S}_L \cup \mathcal{S}_R$);
end
end
end
end
17 **return** $\bigcup_{B \in join_{|q.\text{BGP}|-1}} \mathcal{S}_B$

Algorithm 1: *identifyRelevantSources(Query, QTree)*

Moreover, some queries may actually be answered by a large set of sources, such that a focus on the most important ones becomes important. Both issues suggest to introduce a ranking for sources identified as being relevant for answering the query. There are two different general approaches that could be used to rank sources:

• **external ranking**: ranking based on an independent or externally computed notion of the sources' relevance;

• **cardinality ranking**: ranking based on cardinality.

External ranking may be based on data from external sources (e.g. search engines, requiring additional costly lookups) or may be computed locally. An advantage of cardinality ranking is that we do not need any external data. All necessary information is provided by the QTree buckets that are obtained as a result from the join source selection algorithm. The idea is to estimate the number of results $\mathcal{R}_s$ that each source $s \in \mathcal{S}$ contributes to. The ranks are assigned to sources according to the values of $\mathcal{R}_s$ in descending order.

Each QTree bucket B provides an estimated cardinality c_B and a list of associated sources $\mathcal{S}_B$. To obtain a ranking value for a source (resembling its importance), we could simply assume uniform distribution and assign $c_B/|\mathcal{S}_B|$ to each source of a bucket, while summing up over all buckets. In early tests we recognised that this ranks sources very inaccurately. A simple modification of the QTree, which results in constant space overhead, is to record the cardinality c_B^s for each source contributing to a bucket separately. More specifically, c_B^s estimates the number of results in B that source s contributes to, summed over all joined triples. Thus, $c_B = (\sum_{s \in \mathcal{S}_B} c_B^s)/jl_B$, where jl_B represents the join level of B (i.e., the number of BGPs that have been joined to form one data item in B). This helps to overcome the assumption of a uniform distribution in the bucket. The number of results a source contributes to is determined as:

$$\mathcal{R}_s = \textstyle\sum_B c_B^s$$

Algorithm 1 can be adapted by applying the formulas from lines 3 and 11 separately for each source, while substituting c_B by c_B^s, c_L by c_L^s and c_R by c_R^s.

This is still a rough approximation, but, as we show in Section 5, it indicates the actual importance ranking of sources in a satisfyingly accurate manner. The effect is grounded in probability laws, by which the probability that a source contributes to a fraction of a bucket (the region resulting from the join overlap) increases with its total number of data items in the bucket.

4. DATA SUMMARY CONSTRUCTION & MAINTENANCE

With respect to construction and maintenance, we identify two main tasks, namely i) building an initial version of a QTree (*initial phase*) and ii) expanding the index with new information of sources (*expansion phase*). Once we have an initial version, we can use SPARQL queries to further explore new sources and expand the index in the expansion phase. In the following, we briefly present different approaches for each of the two phases.

4.1 Initial Phase

The initial phase is an important task with high relevance for queries and the expansion of the index. Once the QTree contains the source summaries, SPARQL queries can be evaluated against the index and the resulting relevant documents for query answering can be retrieved from the Web. Users can adjust and influence the completeness of query results and the likelihood of discovering new interesting sources in the expansion phase. If users want to guarantee complete answers, they have to ensure that the QTree contains all relevant sources for the query.

The selection of seed sources influences the ability to discover new and interesting sources in the expansion phase. Let us assume the case that our data summary covers a subgraph containing only few incoming or outgoing links to the rest of the global Linked Data Web. The lack of links to new sources decreases the probability of further extending the index. On the other hand, selecting seed sources which provide many links to other documents increases the chance of discovering new sources. The selection of those well interlinked sources can be done via sampling on a random walk over the Linked Data graph or choosing the top ranked sources of existing datasets.

In general, we identify two different approaches for the initial phase:

• **Pre-fetching** The most obvious approach is to fetch seed sources for the QTree from the Web using a Web crawler. An advantage of this approach is that existing Web crawling systems can be used to gather the seed URIs. The QTree can be adjusted wrt. answer completeness and expansion likeliness by specifying the crawl scope. In particular, random walk strategies generally lead to representative samples of networks and thus result in seed sources that could serve as good entry points to further discover interesting sources [12]. The quality of query answers will depend on the selection of the seed sources and depth/exhaustiveness of the crawl.

• **SPARQL queries** The second approach is starting with an empty QTree and using an initial SPARQL query to collect the initial sources for the QTree build. The index is expanded on further queries; cf. next subsection. Given a SPARQL query, an agent iteratively fetches the content of the URIs selected from bound variables of the query. At least

one dereferenceable URI in the SPARQL query is required as a starting point. Thus, this may be regarded as starting with the plain DL approach mentioned in Section 2.

The decision which strategy to choose strongly depends on the application scenario and has to be chosen accordingly.

4.2 Expansion Phase

The second important phase is the expansion of the QTree index. Given a SPARQL query, it is very likely that the initialised QTree may contain information about dereferenceable URIs that are not (yet) indexed. In this case, the QTree should be updated with the newly discovered URIs to increase the completeness of answer sets for the next time a query is executed. Further, we distinguish between pushing or pulling sources into the QTree:

• **Push of sources** is a passive approach to get new data indexed into the QTree. With passive expansion we refer to all methods that involve users or software agents notifying the QTree about new sources. This can be done by either a service similar to search engines' ping services[4] or by submitting the document directly.

• **Pull of sources** is an active approach to index new data from the Web. One way to achieve this is to perform lazy fetching during query execution. Lazy fetching refers to the process of dereferencing all new URIs needed to answer a query. This particularly fits well with an initial phase based on SPARQL queries, as outlined above. The completeness of queries and the possibility of expanding the QTree with new sources depends on the initial query and can be expected to increase gradually with more queries.

The latter sounds appealing since it solves the cold-start problem elegantly, by performing a plain DL approach on the first query and successively expanding the QTree with more relevant sources. Note that this expansion could be interleaved with prefetching one or two rounds further at each new query, thus accelerating the expansion of the QTree.

Although construction and maintenance are important issues that have to be dealt with in general, we neglect this issue for the remainder of this paper and instead focus on the problem of source selection.

5. EVALUATION

We now present experiments performed on a fixed crawl. On the basis of a set of generated sample queries, we evaluate the performance for determining relevant sources on the QTree and the time elapsed to evaluate the query in memory. Accuracy and quality of the source selection are evaluated on the basis of a benefit measure. Most important for evaluating the practicability of the approach is to measure the impact of source ranking. We also simulate the DL approach and compare it to our method. As the focus of this work is on query processing, we only include basic measurements for index build time; we use the on-disk storage space requirements as a proxy for use of main memory.

We expect the QTree approach to be a lightweight but efficient and effective method to limit the search for query answers to only a subset of relevant sources. However, due to its approximate character, source selection cannot be absolutely accurate. For this we expect the introduced ranking to be a well-suited method for directing search to the most relevant sources. In comparison to the DL approach, our method should be capable of handling more types of queries in reasonable time.

[4]such as for instance `http://pingthesemanticweb.com/` or Sindice [21]

5.1 Setup

Using a breadth first crawl of depth four starting at Tim Berners-Lee's FOAF file[5], we collected about 3 million triples from about 16,000 sources. The data set represents a heterogeneous and well-linked collection of documents hosted on various domains and with different numbers of RDF triples. Most of the sources are manually generated by Semantic Web affiliated users and URIs are reused among documents (e.g., DBpedia or publication/conference URIs). All experiments are performed on a local copy of the gathered data using Java 1.5 and a maximum of 3 GB main memory.

We experimented with queries corresponding to two general classes. The first class of sample query is star-shaped queries with one variable at the subject position. The second type of query is path queries with join variables at subject and object positions. Figure 5 shows abstract representations of these query classes. The query classes of choice are generally understood to be representative for real-world use cases and are also used to evaluate other RDF query systems (e.g., [20]).

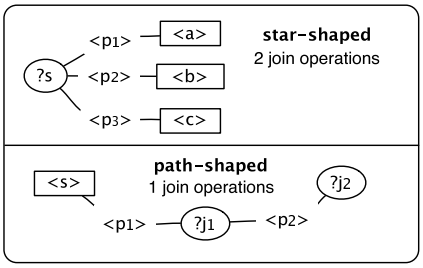


Figure 5: Abstract illustration of used query classes

The star-shaped queries were generated by randomly picking a subject from the input data and arbitrarily selecting distinct outgoing links. Then, we substituted the subject in each BGP with a variable. Path queries were generated using a random walk approach. We randomly chose a subject and performed a random walk of pre-defined depth to select object URIs. The result of such a random walk was transformed into a path-shaped join by replacing the connecting nodes with variables.

Using these approaches, we generated from the data 100 queries for each query class containing one, two or three join operations. We use `P-`n to denote path queries with n join operations and `S-`n to denote star-shaped queries with n join operations. `BGP` refers to queries containing only one BGP and no joins. The figures show averages for all 100 queries in a set. Error bars, if shown, represent minimal and maximal values measured over all tests.

5.2 Results

Next, we present the results of our evaluation, starting with results for index construction. The measured time to insert one triple into the QTree is 4ms on average. The final

[5]`http://www.w3.org/People/Berners-Lee/card`

QTree requires a disk size of around 22 MB in serialised form. As the original data is of size 561 MB, this corresponds to a compression ratio of 96%. In the following, we present the results of four different evaluation aspects: quality of source selection, impact of ranking, query execution time, and comparision with other approaches, and finally discuss the results.

5.2.1 Quality of Source Selection

First, we show the quality achieved for source selection. Based on the total number of sources T in the data, the number of estimated sources E and the number of sources R that are actually needed to answer a query, we calculate the benefit $1.0 - \frac{E}{T}$ for all queries. The benefit measures the number of sources that can be skipped in the query process, compared to the naïve approach of simply querying all known sources. In other words, the benefit gives an idea pf how much we save: i.e., how many sources we can discard from querying without missing results. Figure 6 shows the benefit for various query types. We observe a benefit of above 80% for the star-shaped queries, while for path queries we achieve benefits of about 20%, 40% and 60%. The high benefit shows that our approach is very well suited to prune the search space of all sources. The difference between query classes is due to the fact that star queries are answered by significantly fewer sources than path queries, which usually span a large number of documents. Thus, the benefit for path queries cannot be as high as for star queries. However, the number of possibly relevant sources can still be in the thousands. This highlights the importance of an accurate source ranking.

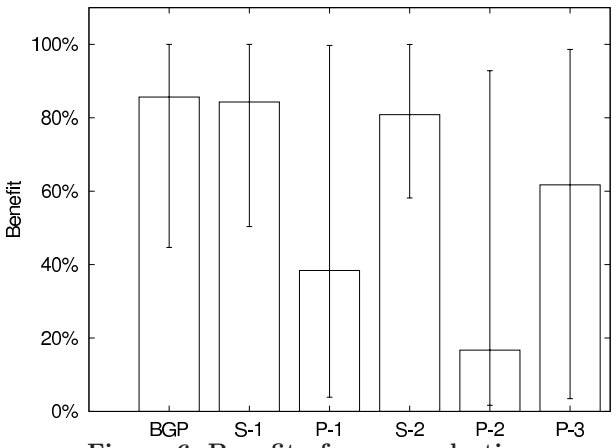


Figure 6: Benefit of source selection

5.2.2 Impact of Ranking

An accurate ranking scheme is mandatory in the presence of a huge number of relevant sources. To show the impact of the ranking, we measured how many result triples we can determine and how many queries we can completely answer when querying only top-k ranked sources. We show results for reasonable values of k, namely 10, 50, 100 and 200. Figure 7 and 8 illustrate the results of this test. In addition, Figure 9 shows the average maximal k that would be required to answer a query completely (i.e., to achieve 100% in Figure 7). The figure further shows the number of actual relevant sources. We can conclude that the introduced ranking is powerful and important for practical applications. The recall values for the plots in Figure 7 are above 50% for 4 out of 7 tests with the top-200 sources. Inspecting the ratio of completely answered queries for the query types, we observe that the path queries dominate the star-shaped queries. This is a nice complement to the higher benefit values for star-shaped queries. Figure 9 shows that the absolute error in the number of selected sources increases with the complexity of queries.

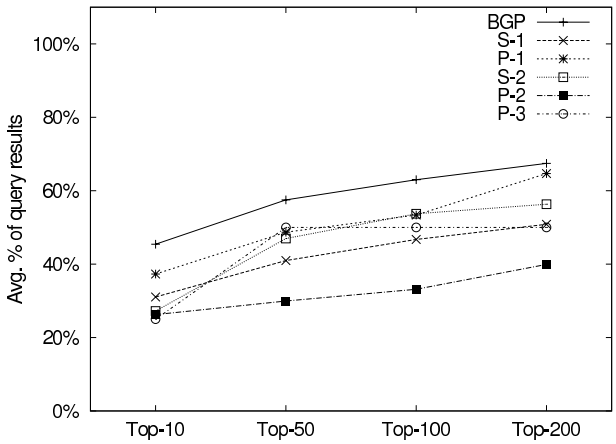


Figure 7: Impact of ranking, recall of triples

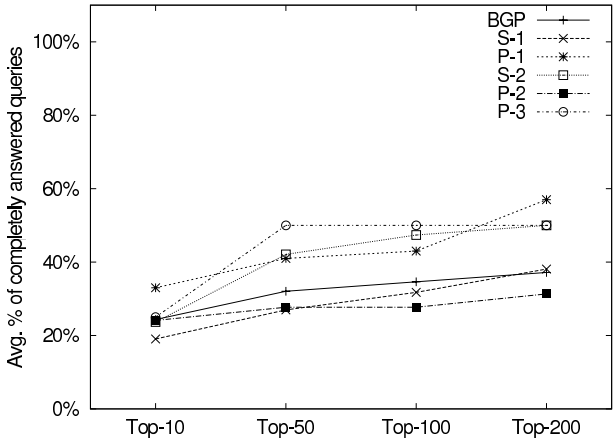


Figure 8: Impact of ranking, answer completeness

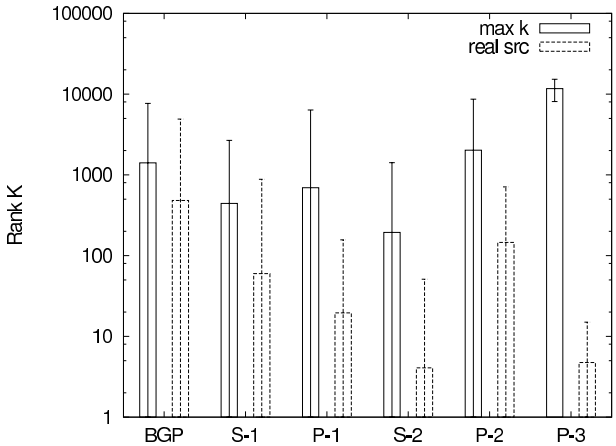


Figure 9: Impact of ranking, maximal k

	BGP	S-1	P-1	S-2	P-2	P-3
Average	32.8%	20%	-	9.64%	-	-
Maximum	100%	39.8%	-	27.8%	-	-

Table 1: Completeness of results with the DL approach

5.2.3 Query Execution Time

A crucial aspect besides quality and benefit of the source selection is runtime performance, i.e., the actual time needed to answer queries. Figure 10 shows the average time required to estimate relevant sources (*qtree*) and to actually evaluate the query afterwards on the content stored in memory (*query*). The average query time for all queries is below 10 seconds, with some outliers of maximum 100 seconds. This difference in the query times results from the number of relevant sources, which is in parts very high (according to the QTree, but also the actual number of relevant sources for some queries). Similar times can be observed for source selection on the QTree; the difference here is also due to the number of buckets that have to be checked while answering single BGPs on the QTree, as query times increase with the number of buckets. The shown query times underline the applicability and practicability of our approach for a real-world application.

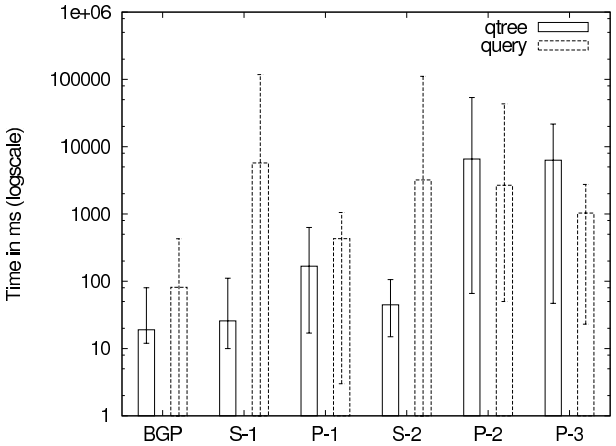


Figure 10: Query time

5.2.4 Comparison with Other Approaches

Finally, we compare our proposed solution with an alternative approach, namely the DL approach. We implemented a local generalised version of the algorithm for a fair comparison with our proposed solution. For comparison we emulate the approach using the crawled local data. We cannot expect the results to be completely accurate since since the DL approach performs, by design, live HTTP lookups. Despite this difference, an evaluation based on crawl data reflects the general limitations of the DL approach. Table 1 shows that the DL approach is capable of returning results only for star-shaped queries with less than 2 joins, for path queries the DL approach returned no results.

5.3 Discussion

The evaluation shows that our novel approach is very promising and practical for efficiently querying the Linked Data Web. The problems of state-of-the-art solutions can be eliminated successfully by the use of memory-efficient index structures such as the QTree. As expected, this is only practical if an accurate ranking is applied. We were able to show that even a straightforward cardinality-based ranking is well suited to achieve this task. Our proposed solution is applicable to real-world scenarios, given the presented index and query times and the precision and impact of the top-k ranking. A client, able to perform multithreaded lookups and set up with an appropriate timeout for fetching the content of the estimated sources, can answer queries with live results in less then a minute using an index of 4% size of the original data. Almost all of our expectations were met by the evaluation. However, the precision of the QTree index is slightly below our expectations and can benefit from optimisations. In summary, the proposed approach represents a novel, efficient and effective way of supporting source selection for live queries over the Linked Data Web. It is in a state ready for real-world applications, although the very promising results can still be tuned.

6. RELATED WORK

An implementation of the naïve Data Lookup approach – i.e., iterative query processing with dereferencing bound URIs – has been recently presented by Hartig et al. [10]. As already sketched in Section 4, we believe our approach can be viewed as fruitfully expanding and generalising the straightforward approach towards more complete and versatile query answering over Linked Data.

Database systems have exploited the idea of capturing statistics about data for many years by using histograms [16], primarily for selectivity and cardinality estimates over local data.

The majority of work on distributed query optimisation assumes a relatively small number of endpoints with full query processing functionality rather than a possibly huge number of flat file containing small amounts of data. Stuckenschmidt et al. [26] proposed an index structure for distributed RDF repositories based on schema paths (property chains) rather than on statistical summaries of the graph-structure of the data. RDFStats [18] aims at providing statistics for RDF data that can be used for query processing and optimisation over SPARQL endpoints. Statistics include histograms, covering e.g., subjects or data types, and estimates cardinalities of selected BGPs and example queries. The Vocabulary of Interlinked Datasets (voiD)[6] is a format for encoding and publishing statistics such as basic histograms in RDF. The QTree contains more complete selectivity estimates for all BGPs of distributed Linked Data sources and the ability to estimate selectivity of joins.

A recent system using B^+-trees to index RDF data is RDF-3X [20]. To answer queries with variables in any position of an RDF triple, RDF-3X holds indexes for querying all possible combinations of subject, predicate and object – an idea introduced in [9]. RDF-3X uses sophisticated join optimisation techniques based on statistics derived from the data. In contrast to our work, the approach uses a different data structure for the index and focuses on centralised RDF stores rather than distributed Linked Data sources.

Peer-to-peer systems (P2P) leverage statistical data for source selection using so-called routing indexes. Crespo et al. [5] introduced the notion of routing indexes in P2P systems as structures that, given a query, return a list of inter-

[6] `http://rdfs.org/ns/void`

esting neighbours (sources) based on a data structure conforming to lists of counts for keyword occurrences in documents. Based on this work, other variants of routing indexes have been proposed, e.g., based on one-dimensional histograms [22], Bloom Filters [23], bit vectors [19], or the QTree. A common feature across these systems is to use a hash function to map string data to a numerical data space. In contrast to our work, the focus of query optimisation in P2P systems is to share load among multiple sites and on local optimisation based on routing indexes.

7. CONCLUSION & FUTURE WORK

We have presented an approach for evaluating queries over RDF published as Linked Data, based on an index structure which summarises the content of data sources. We have shown how the index structure can be used to select relevant sources for conjunctive query answering, and how to process joins over relevant sources with an optional prioritisation via ranking. We have discussed strategies for constructing such data summaries from a static dataset or dynamically during query evaluation, and presented experimental results and discussion of our approach on synthetically generated queries over a Web crawl from 16k sources consisting of 3m RDF triples. We have shown that our approach is able to handle more expressive queries and return more complete results to queries compared to previous approaches.

While our initial results are promising, there remain several issues and future directions to explore. Restricting the number of lookups via cardinality ranking reduces the overall processing time in our current approach. However, we would like to investigate what types of ranking could be used to further improve the accuracy of the lookups. In addition, performing reasoning over the collected data would allow for returning consistent results adhering to the specified semantics. The current work describes the general applicability of approximate index structures for query processing, however, future work will have to study approaches for index creation and maintenance in more detail. We plan to deploy a query engine with a populated QTree for public user queries and investigate how a QTree purely built on real user queries evolves. Last but not least, we should highlight that QTrees are also applicable in a fully decentralised distributed querying scenario where peers are able to process and forward queries themselves.

8. REFERENCES

[1] T. Berners-Lee. Linked data, July 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[2] D. Brickley, L. Miller. FOAF Vocabulary Spec. 0.91, 2007. `http://xmlns.com/foaf/spec/`.

[3] G. Cheng, Y. Qu. Searching linked objects with falcons: Approach, implementation and evaluation. *JSWIS*, 5(3):49–70, 2009.

[4] K. G. Clark, L. Feigenbaum, E. Torres. SPARQL protocol for RDF, Jan. 2008. W3C Rec., `http://www.w3.org/TR/rdf-sparql-protocol/`.

[5] A. Crespo, H. Garcia-Molina. Routing indices for peer-to-peer systems. *ICDCS '02*, p.23–32, 2002.

[6] M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, E. Motta. Characterizing knowledge on the semantic web with watson. *EON'07*, p.1–10, 2007.

[7] R. Goldman, J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. *VLDB'97*, p.436–445, 1997.

[8] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD '84*, p.47–57, 1984.

[9] A. Harth, S. Decker. Optimized index structures for querying RDF from the web. *3rd Latin American Web Congress*, p.71–80, 2005.

[10] O. Hartig, C. Bizer, J.-C. Freytag. Executing sparql queries over the web of linked data. *ISWC'09*, 2009.

[11] D. Heimbigner, D. McLeod. A federated architecture for information management. *ACM Trans. Inf. Syst.*, 3(3):253–278, 1985.

[12] M. R. Henzinger, A. Heydon, M. Mitzenmacher, M. Najork. Measuring index quality using random walks on the web. *Computer Networks*, 31(11-16):1291–1303, 1999.

[13] A. Hogan, A. Harth, J. Umbrich, S. Decker. Towards a scalable search and query engine for the web. *WWW'07*, p.1301–1302, 2007.

[14] K. Hose, M. Karnstedt, A. Koch, K. Sattler, D. Zinn. Processing Rank-Aware Queries in P2P Systems. *DBISP2P'05*, p.238–249, 2005.

[15] K. Hose, D. Klan, K. Sattler. Distributed Data Summaries for Approximate Query Processing in PDMS. *IDEAS '06*, p.37–44, 2006.

[16] Y. Ioannidis. The History of Histograms (abridged). *VLDB '03*, p.19–30, 2003.

[17] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, Dec. 2000.

[18] A. Langegger, W. Wöß. RDFstats - an extensible RDF statistics generator and library. *8th Int'l Workshop on Web Semantics, DEXA*, 2009.

[19] M. Marzolla, M. Mordacchini, S. Orlando. Tree Vector Indexes: Efficient Range Queries for Dynamic Content on Peer-to-Peer Networks. *PDP'06*, p.457–464, 2006.

[20] T. Neumann, G. Weikum. RDF-3X: a RISC-style Engine for RDF. *VLDB Endow.*, 1(1):647–659, 2008.

[21] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, G. Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *JMSO*, 3(1), 2008.

[22] Y. Petrakis, G. Koloniari, E. Pitoura. On Using Histograms as Routing Indexes in Peer-to-Peer Systems. *DBISP2P '04*, p.16–30, 2004.

[23] Y. Petrakis and E. Pitoura. On Constructing Small Worlds in Unstructured Peer-to-Peer Systems. *EDBT Workshops*, p.415–424, 2004.

[24] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF, Jan. 2008. W3C Rec., `http://www.w3.org/TR/rdf-sparql-query/`.

[25] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. *ESWC'08*, p.524–538, Tenerife, Spain, 2008.

[26] H. Stuckenschmidt, R. Vdovjak, G.-J. Houben, J. Broekstra. Index structures and algorithms for querying distributed RDF repositories. *WWW'04*, p.631–639, 2004.

[27] D. Zinn. Skyline Queries in P2P Systems. *Diploma Thesis*, TU Ilmenau, 2004.

[P2] Sebastian Speiser and Andreas Harth. Integrating Linked Data and Services with Linked Data Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, pages 170–184. 2011.

Integrating Linked Data and Services with Linked Data Services *

Sebastian Speiser and Andreas Harth

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany
lastname@kit.edu

Abstract. A sizable amount of data on the Web is currently available via Web APIs that expose data in formats such as JSON or XML. Combining data from different APIs and data sources requires glue code which is typically not shared and hence not reused. We propose Linked Data Services (LIDS), a general, formalised approach for integrating data-providing services with Linked Data, a popular mechanism for data publishing which facilitates data integration and allows for decentralised publishing. We present conventions for service access interfaces that conform to Linked Data principles, and an abstract lightweight service description formalism. We develop algorithms that use LIDS descriptions to automatically create links between services and existing data sets. To evaluate our approach, we realise LIDS wrappers and LIDS descriptions for existing services and measure performance and effectiveness of an automatic interlinking algorithm over multiple billions of triples.

1 Introduction

The trend towards publishing data on the Web is gaining momentum, particularly spurred by the Linking Open Data (LOD) project[1] and several government initiatives aimed at publishing public sector data. Data publishers often use Linked Data principles [3]. which leverage established Web standards such as Uniform Resource Identifiers (URIs), the Hypertext Transfer Protocol (HTTP) and the Resource Description Framework (RDF) [4]. Data providers can easily link their data to data from third parties via reuse of URIs. The LOD project proves that the Linked Data approach is, in principle, capable of integrating data from a large number of sources. However, there is still a lot of data residing in silos that could be beneficially linked with other data, but will not be published as a fully materialised knowledge base. Reasons include:

– data is constantly changing, e.g., stock quotes or sensor data can have update intervals below one second;

* This paper is an extension of our previous work [1, 2]. We have extended the work with a formal definition of service descriptions, an evaluation of the performance and effectiveness of the proposed methods – including the implementation of several Linked Data Services – and an extensive overview of related work.

[1] http://linkeddata.org/

– data is generated depending on possibly infinite different input data, e.g., the distance between two geographical points can be specified with arbitrary precision;
– the data provider does not want arbitrary access to the data, e.g., prices of flight tickets may be only available for specific requests in order to maintain the possibility for price differentiation.

Such data is commonly provided via Web APIs or services, in the following also called data or information services, as they provide a restricted view on a possibly implicit data set. APIs are often based on Representational State Transfer (REST) principles [5], use HTTP as transport protocol and pass parameters as name/value pairs in the URI query string. Currently deployed Web APIs return data as JSON or XML, which requires glue code to combine data from different APIs.

There are useful examples for the integration of information services and Linked Data. Linked Data interfaces for services have been created, e.g., in form of the book mashup [6] which provides RDF about books based on Amazon's API, or twitter2foaf, which encodes a Twitter follower network of a given user based on Twitter's API. However, the interfaces are not formally described and thus the link between services and data has to be established manually or by service-specific algorithms. For example, to establish a link between person instances (e.g., described using the FOAF vocabulary[2]) and their Twitter account, one has to hard-code which property relates people to their Twitter username and the fact that the URI of the person's Twitter representation is created by appending the username to `http://twitter2foaf.appspot.com/id/`.

Vast amounts of idle data can be brought to the Semantic Web via a standardised method for creating Linked Data interfaces to services. The method should incorporate formal service descriptions that enable (semi-)automatic service discovery and integration. We present such an approach for what we call LInked Data Services (LIDS). Specifically, we present the following contributions:

– an access mechanism for LIDS interfaces based on generic Web architecture principles (URIs and HTTP) (Section 3);
– a generic lightweight data service description formalism, instantiated for RDF and SPARQL graph patterns (Section 4);
– an algorithm for linking existing data sets using LIDS (Section 5)

In Section 6 we describe the creation of LIDS for existing services, and present the results of an experiment measuring performance and effectiveness of the approach. The experiment interlinks the 2010 Billion Triple Challenge data set with a geographic LIDS. We relate our approach to existing work in Section 7 and conclude with Section 8.

[2] `http://xmlns.com/foaf/0.1/`

2 Preliminaries

In the following we shortly present the basics for our work, namely: data services, and RDF.

2.1 Data Services

Our notion of data services is as follows:

> Data services return data dynamically derived (i.e., during service call time) from supplied input parameters. Data services neither alter the state of some entity nor modify data. In other words, data services are free of any side effects. They can be seen as data sources providing information about some entity, when given input in the form of a set of name/value pairs. The notion of data services include Web APIs and REST-based services providing output data in XML or JSON.

Data services are related to Web forms or the "Deep Web" [7], but take and provide data rather than free text or documents. For example, the GeoNames `findNearbyWikipedia` service relates given latitude/longitude parameters to Wikipedia articles describing geographical features that are nearby.

API	Format	Description
GeoNames	XML, JSON	Functions include besides others: (i) find the nearest GeoNames feature to a given point and (ii) link a geographic point to resources from DBpedia that are nearby URL: `http://www.geonames.org/`
Google GeoCoding API	XML, JSON	Provides latitude and longitude for a given street address. URL: `http://code.google.com/apis/maps/`
Twitter API	XML, JSON, RSS, Atom	Various functions, giving access to Twitter users, follower networks, and tweets. URL: `http://dev.twitter.com/`

Table 1. Example data-providing services.

Example 1. In Table 1, we list some popular data-providing services. Taking the Google GeoCoding API, to get the geographical coordinates for Karlsruhe, we retrieve the URI `http://maps.googleapis.com/maps/api/geocode/json?address=Karlsruhe&sensor=false`, with the following (abbreviated) result:

```
{ "status": "OK",
  "results": [ {
  ...
```

```
  "formatted_address": "Karlsruhe, Germany",
  ...
  "geometry": {
    "location": {
      "lat": 49.0080848,
      "lng": 8.4037563
    },
    ...
  } } ] }
```

Using the retrieved coordinates, we can build the URI for calling the GeoNames service to find Wikipedia articles about things, that are nearby Karlsruhe: `http://ws.geonames.org/findNearbyWikipedia?lat=49.0080848&lng=8.4037563`. The (abbreviated) result is the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
  <entry>
    <lang>en</lang>
    <title>Federal Constitutional Court of Germany</title>
    ...
    <lat>49.0125</lat>
    <lng>8.4018</lng>
    <wikipediaUrl>...</wikipediaUrl>
    ...
  </entry>
  <entry>
    ...
  </entry>
</geonames>
```

This simple example shows that integrating data from several (in this case only two) services is difficult for the following reasons:

- different serialisation formats are used (e.g., JSON, XML);
- entities are not represented explicitly, and are thus difficult to identify between different services. For example, the geographical point returned by the GeoCoding API does not occur in the output of the GeoNames service. Therefore it is not possible to link the results based on the service outputs alone, but only with service-specific gluing code.

2.2 RDF and Basic Graph Patterns

In contrast to XML or JSON, the Resource Description Framework (RDF) is a graph-based data format which allows for easy integration of data from multiple sources. We now introduce basic RDF notions later reused in the paper; cf. [**?**].

Let U, B, L, V be disjoint infinite sets of URIs, blank nodes, literals and variables.

Definition 1. *(Triple) A triple $t = (s, p, o)$ is a tuple of length three, $t \in (U \cup B) \times U \times (U \cup B \cup L)$. We often write t as* `s p o`*, where* `s` *is called the subject,* `p` *the predicate and* `o` *the object.*

Definition 2. *(RDF Graph) An RDF graph r is a finite set of triples.*

We often write a set of triples by separating triples by `.` (a dot). To be able to query graphs, we introduce the notion of triple pattern which can include variables.

Definition 3. *(Triple Pattern) A triple pattern $t \in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ abstracts from single triples by allowing variables in every position.*

Definition 4. *(Basic Graph Pattern (BGP) and Conjunctive Query (CQ)) A BGP is a finite set of triple patterns. A conjunctive query $CQ = (X, T)$ consists of a head, i.e. a set of variables $X \subset V$, and a body, i.e. a BGP T.*

Let M be the set of all function $\mu : U \cup L \cup V \to U \cup L$, s.t. μ is the identity for constants, i.e. $\forall a : (a \in U \cup L \to \mu(a) = a)$. As an abbreviation we also apply a function $\mu \in M$ to a triple pattern $t = p(t_1, \ldots, t_n)$ $(\mu(t) = p(\mu(t_1), \ldots, \mu(t_n)))$, and to a BGP T $(\mu(T) = \{\mu(t) \mid t \in T\})$.

Definition 5. *(Variable Binding) A function $\mu \in M$ is a variable binding for a conjunctive query $CQ = (X, T)$ and a RDF graph r, if $\mu(T) \subseteq r$. We denote the set of all mappings for a CQ and a graph as $\mathcal{M}_{CQ}(r) = \{\mu \in M | \mu(T) \subseteq r\}$.*

3 Linked Data Services

Linked Data Services provide a Linked Data interface for data services. To make these services adhere to Linked Data principles a number of requirements have to be fulfilled:

- the input for a service invocation with given parameter bindings must be identified by a URI;
- resolving that URI must return a description of the input entity, relating it to the service output data;
- the description must be returned in RDF format.

We call such services *Linked Data Services (LIDS).*

Example 2. Inputs for the LIDS version of the `findNearbyWikipedia` service are entities representing geographical points given by latitude and longitude, which are encoded in the URI of an input entity. Resolving such an input URI returns a description of the corresponding point, which relates it to Wikipedia articles which are nearby.

Defining that the URI of a LIDS call identifies an input entity is an important design decision. Compared to the alternative – directly identifying output entities with service call URIs – identifying input entities has the following advantages:

– the link between input and output data is made explicit;
– one input entity (e.g., a geographical point) can be related to several results (e.g., Wikipedia articles);
– the absence of results can be easily represented by an description without further links;
– the input entity has a constant meaning although data can be dynamic (e.g., the input entity still represents the same point, even though a subsequent service call may relate the input entity to new or updated Wikipedia articles).

More formally we characterise a LIDS by:

– Linked Data Service endpoint: ep, an HTTP URI.
– Local identifier i for the input entity of the service.
– Inputs X_i: names of parameters.

The URI of a service call for a parameter assignment μ (mapping X_i to corresponding values) is constructed in the following way (where addition is understood as string concatenation and subtraction removes the corresponding suffix if it matches):

$$uri(ep, X_i, \mu) = ep + \texttt{"?"} + \sum_{x \in X_i} (x + \texttt{"="} + \mu(x) + \texttt{"\&"}) - \texttt{"\&"}$$

Additionally we introduce an abbreviated URI schema that can be used if there is only one required parameter (i.e. $|X_i| = 1, X_i = \{x\}$):

$$uri(ep, X_i, \mu) = ep + \texttt{"/"} + \mu(x)$$

Please note that the above definition coincides with typical Linked Data URIs. The input entity described by the output of a service call is defined as $inp(ep, X_i, \mu, i) = uri(ep, X_i, \mu) + \texttt{"\#"} + i$.

Example 3. We illustrate the principle using the openlids.org wrapper for GeoNames[3] `findNearbyWikipedia`. The wrapper is a LIDS, defined by:

– endpoint $ep =$ `gw:findNearbyWikipedia`;
– local identifier $i =$ "*point*";
– inputs $X_i = \{$"*lat*","*lng*"$\}$.

For a binding $\mu = \{lat \mapsto 49.01, lng \mapsto 8.41\}$ the URI for the service call is `gw:findNearbyWikipedia?lat=49.01&lng=8.41` and returns the following description:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
gw:findNearbyWikipedia?lat=49.01&lng=8.41#point
    foaf:based_near dbpedia:University_of_Karlsruhe_%28TH%29;
    foaf:based_near dbpedia:Federal_Constitutional_Court_of_Germany;
    foaf:based_near dbpedia:Federal_Court_of_Justice_of_Germany;
    foaf:based_near dbpedia:Wildparkstadion;
    foaf:based_near dbpedia:Karlsruhe.
```

[3] `http://km.aifb.kit.edu/services/geowrap/`, abbreviated as `gw`.

4 Describing Linked Data Services

In this section, we define an abstract model of LIDS descriptions.

Definition 6. *(LIDS Description) A LIDS description consists of a tuple* (ep, CQ_i, T_o, i) *where ep denotes the LIDS endpoint,* $CQ_i = (X_i, T_i)$ *a conjunctive query to specify the input to the service,* T_o *a basic graph pattern describing the output data of the service, and i the local identifier for the input entity.*

The meaning of *ep* and X_i were already explained in the previous section. We define X_i to be the head of a conjunctive query, whose body specifies the required relation between the input parameters. T_o specifies the minimum output that is returned by the service for valid input parameters. More formally:

- $\mu \in M$ is a valid input, if $\mu \in M_{CQ_i}(r)$, where r is the implicit RDF graph given by all Linked Data;
- for a valid μ, resolving $uri(ep, X_i, \mu)$ returns a graph $D_o \supseteq \{T' \subseteq D_{impl} \mid \exists \mu \in M : \mu(i) = E_s \wedge \mu(T_o) = T'\}$, where D_{impl} is the implicit, potentially infinite data set representing the information provided by the LIDS.

Example 4. We describe the `findNearbyWikipedia` openlids.org wrapper service as (ep, CQ_i, T_o, i) with:
$ep =$ `gw:findNearbyWikipedia`
$CQ_i =$ ({lat,lng}, { `?point geo:lat ?lat . ?point geo:long ?lng` })
$T_o =$ {`?point foaf:based_near ?feature`}
$i = point$

4.1 Relation to Source Descriptions in Information Integration Systems

Note that the LIDS descriptions can be transformed to source descriptions with limited access patterns, in a Local-as-View (LaV) data integration approach [8]. With LaV, the data accessible through a service is described as a view in terms of a global schema. The variables of a view's head predicate that have to be bound in order to retrieve tuples from the view are prefixed with a \$. For a LIDS description (ep, CQ_i, T_o, i), we can construct the LaV description:

$$ep(\$I_1, \ldots, \$I_k, O_1 \ldots, O_m) \texttt{ :- } p_1^i(\ldots), \ldots, p_n^i(\ldots), p_1^o(\ldots), \ldots, p_l^o(\ldots).$$

Where $CQ_i = (X_i, T_i)$, $X_i = \{I_1, \ldots, I_k\}$, $T_i = \{(s_1^i, p_1^i, o_1^i), \ldots, (s_n^i, p_n^i, o_n^i)\}$, $T_o = \{(s_1^o, p_1^o, o_1^o), \ldots, (s_l^o, p_l^o, o_l^o)\}$, and $vars(T_o) \setminus vars(T_i) = \{O_1, \ldots, O_m\}$.

We propose for LIDS descriptions the separation of input and output conditions for three reasons: (i) the output of a LIDS corresponds to an RDF graph as described by the output pattern, not to tuples as it is common in LaV approaches, (ii) it is easier to understand for users, and (iii) it is better suited for the interlinking algorithm as shown in Section 5.

4.2 Describing LIDS using RDF and SPARQL Graph Patterns

In the following we present how LIDS descriptions can be represented in RDF, thus enabling that LIDS descriptions can be published as Linked Data. The basic format is as follows (unqualified strings consisting only of capital letters are placeholders and explained below):

```
@prefix lids: <http://openlids.org/vocab#>

LIDS a lids:LIDS;
     lids:lids_description [
         lids:endpoint ENDPOINT ;
         lids:service_entity ENTITY ;
         lids:input_bgp INPUT ;
         lids:output_bgp OUTPUT ;
         lids:required_vars VARS
      ] .
```

The RDF description is related to our abstract description formalism in the following way:

- `LIDS` is a resource representing the described Linked Data service;
- `ENDPOINT` is a URI corresponding to ep;
- `ENTITY` is the name of the entity i;
- `INPUT` and `OUTPUT` are basic graph patterns encoded as a string using SPARQL syntax. `INPUT` is mapped to T_i and `OUTPUT` is mapped to T_o.
- `VARS` is a string of required variables separated by blanks, which is mapped to X_i.

From this mapping, we can construct an abstract LIDS description $(ep, (X_i, T_i), T_o, i)$ for the service identified by `LIDS`.

Example 5. In the following we show the RDF representation of the formal LIDS description from Example 4:

```
:GeowrapNearbyWikipedia a lids:LIDS;
  lids:lids_description [
     lids:endpoint
       <http://km.aifb.kit.edu/services/geowrap/findNearbyWikipedia>;
     lids:service_entity "point" ;
     lids:input_bgp "?point a Point . ?point geo:lat ?lat .
                                      ?point geo:long ?long" ;
     lids:output_bgp "?point foaf:based_near ?feature" ;
     lids:required_vars "lat long"
  ] .
```

In future, we expect a standardised RDF representation of SPARQL, which does not rely on string encoding of basic graph patterns. One such candidate is the

SPIN SPARQL Syntax[4], which is part of the SPARQL Inferencing Notation (SPIN)[5]. We are planning to reuse such a standardised RDF representation of basic graph patterns and variables in future versions of the LIDS description model.

5 Algorithm for Interlinking Data with LIDS

In the following, we describe how existing data sets can be automatically enriched with links to LIDS, which can happen in different settings. Consider for example:

- processing of a static data set, inserting links to LIDS and storing the new data;
- an endpoint that serves data (e.g., a Linked Data server), and dynamically adds links to LIDS;
- a data browser that locally augments retrieved data with data retrieved from LIDS.

We present an algorithm that, based on a fixed local dataset, determines and invokes the appropriate LIDS and adds the output to the local dataset.

Given an RDF graph r and a LIDS description $l = (ep, CQ_i, T_o)$ the following formula defines a set of entities in r and equivalent entities that are inputs for the LIDS (i is determined from T_i and T_o and $+$ is again string concatenation):

$$equivs_{r,l} = \Big\{ \big(\mu(i), uri(ep, X_i, \mu) + \text{"\#"} + i\big) \mid \mu \in \mathcal{M}_{CQ_i}(r) \} \Big\}.$$

The obtained equivalences can be either used to immediately resolve the LIDS URIs and add the data to r, or to make the equivalences explicit in r, for example, by adding the following triples to r:

$$\big\{ x_1 \texttt{ owl:sameAs } x_2 \mid (x_1, x_2) \in equivs_{r,l} \big\}.$$

Based on the services shown in Figure 1 together with descriptions, we illustrate the algorithm using the following example: consider as starting point an entity URI (e.g., an entity `#aifb`), which, when visited, returns an RDF graph with latitude and longitude properties:

```
#aifb
    rdfs:label "AIFB - Building 11.40";
    geo:lat "49.01";
    geo:long "8.41".
```

In the first step, the data is matched against the available LIDS descriptions (for brevity we assume a static set of LIDS descriptions) and a set of bindings are derived. Further processing uses the GeoNames LIDS which accepts latitude/longitude as input. After constructing a URI which represents the service entity, an equivalence (`owl:sameAs`) link is created between the original entity `#aifb` and the service entity:

[4] http://spinrdf.org/sp.html
[5] http://spinrdf.org/

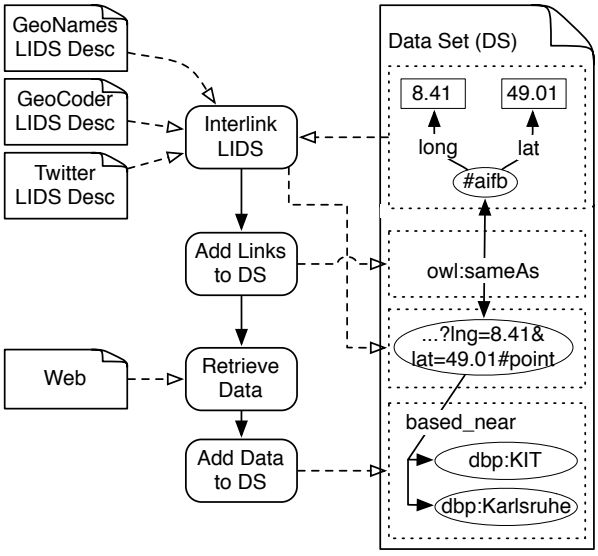


Fig. 1. Interlinking example for GeoNames LIDS

```
#aifb owl:sameAs
       gw:findWikipediaNearby?lat=49.01&long=8.41#point.
```

Next, the data from the service entity URI can be retrieved, to obtain the following data:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
gw:findWikipediaNearby?lat=49.01&long=8.41#point
       foaf:based_near foaf:based_near dbpedia:Wildparkstadion;
       foaf:based_near dbpedia:Karlsruhe.
...
```

Please observe that by equating the URI from the input data with the LIDS entity URI, we essentially add the returned `foaf:based_near` statements to `#aifb`. Should the database underlying the service change, a lookup on the LIDS entity URI returns the updated data which can then be integrated. As such, entity URIs can be linked in the same manner as plain Linked Data URIs.

6 Evaluation of Performance and Effectiveness

We first present several LIDS services which we have made available, and then cover the evaluation of performance and effectiveness of the presented algorithm. Source code and test data for the implementation of the interlinking algorithm, as well as other general code for handling LIDS and their descriptions can be

found online[6]. All experiments were conducted on a 2.4 GHz Intel Core2Duo laptop with 4 GB of main memory.

6.1 Implemented LIDS Services

In this section, we show how we applied the LIDS approach to construct publicly available Linked Data interfaces for selected existing services.

The following services are hosted on Google's App Engine cloud environment. The services are also linked on `http://openlids.org/` together with their formal LIDS descriptions and further information, such as URIs of example entities.

- GeoNames Wrapper[7] provides three functions:
  - finding the nearest GeoNames feature to a given point,
  - finding the nearest GeoNames populated place to a given point,
  - linking a geographic point to resources from DBpedia that are nearby.
- GeoCoding Wrapper, returning the geographic coordinates of a street address.
- Twitter Wrapper[8] links Twitter account holders to the messages they post.

The effort to produce a LIDS wrapper is typically low. The interface code that handles the service URIs and extracts parameters can be realised by standardised code or even generated automatically from a LIDS description. The main effort lies in accessing the service and generating a mapping from the service's native output to a Linked Data representation. While for some services it is sufficient to write a simple JavaScript wrapper that transforms JSON data into RDF/N3, other services require simple Java procedures or XSLTs transforming output XML data to RDF/XML. Effort is higher for services that map Web page sources, as this often requires session and cookie handling and parsing of faulty HTML code. However, the underlying data conversion has to be carried out whether or not LIDS are used. Following the LIDS principles is only a minor overhead in implementation; adding a LIDS descriptions requires a SPARQL query to describe the service.

6.2 Interlinking Existing Data Sets with LIDS

We implemented a streaming version of the interlinking algorithm shown in Section 5 based on NxParser[9]. For evaluation of the algorithm's performance and effectiveness we interlinked the Billion Triple Challenge (BTC) 2010 data set[10] with the `findNearby` geowrapper. In total the data set consisted of 3,162,149,151 triples and was annotated in 40,746 seconds ($<$ 12 hours) plus about 12 hours for uncompressing the data set, result cleaning, and statistics gathering. In the

[6] `http://code.google.com/p/openlids/`
[7] `http://km.aifb.kit.edu/services/geowrap/`
[8] `http://km.aifb.kit.edu/services/twitterwrap/`
[9] `http://sw.deri.org/2006/08/nxparser/`
[10] `http://km.aifb.kit.edu/projects/btc-2010/`

cleaning phase we filtered out links to the geowrapper that were redundant, i.e., entities that were already linked to GeoNames, including the GeoNames data set itself. The original BTC data contained 74 different domains that referenced GeoNames URIs. Our interlinking process added 891 new domains that are now linked to GeoNames via the geowrap service. In total 2,448,160 new links were added[11]. Many links referred to the same locations, all in all there were links to ca. 160,000 different geowrap service calls. These results show that even with a very large data set, interlinking based on LIDS descriptions is feasible on commodity hardware. Furthermore, the experiment showed that there is much idle potential for links between data sets, which can be uncovered with our approach.

7 Related Work

Our work provides an approach to open up data silos for the Web of Data. Previous efforts in this direction are confined to specialised wrappers, for example the book mashup [6]. Other state-of-the-art data integration systems [9] use wrappers to generate RDF and then publish that RDF online rather than providing access to the services that generate RDF directly. In contrast to these ad-hoc interfaces, we provide a uniform way to construct such interfaces, and thus our work is applicable not only to specific examples but generally to all kinds of data silos. Furthermore, we present a method for formal service description that enables the automatic interface generation and service integration into existing data sets.

SILK [10] can be used to discover links between Linked Data from different sources. Using a declarative language, a developer specifies conditions that data from different sources has to fulfill to be merged, optionally using heuristics in case merging rules can lead to ambiguous results. In contrast, we use Linked Data principles for exposing content of data-providing services, and specify the construction of URIs which can be related to already existing data.

There exists extensive literature about semantic descriptions of Web services. We distinguish between two kinds of works: (i) general semantic Web service (SWS) frameworks, and (ii) stateless service descriptions.

General SWS approaches include OWL-S [11] and WSMO [12] and aim at providing extensive expressivity in order to formalise every kind of Web service, including complex business services with state changes and non-trivial choreographies. The expressivity comes at a price: SWS require complex modeling even for simple data services using formalisms that are not familiar to all Semantic Web developers. In contrast, our approach focuses on simple data services and their lightweight integration with Linked Data.

Most closely related to our service description formalism are works on semantic descriptions of stateless services (e.g., [13–15]). Similar to our approach these solutions define service functionality in terms of input and output conditions. Most of them, except [13], employ proprietary description formalisms. In

[11] Linking data is available online: `http://people.aifb.kit.edu/ssp/geolink.tgz`

contrast, our approach relies on standard SPARQL. Moreover, our work provides the following key advantages: (i) a methodology to provide a Linked Data interface to services, (ii) semi-structured input and output definitions, compared to the static definition of required inputs and outputs in previous approaches.

Norton and Krummenacher propose an alternative approach to integrate Linked Data and services, so-called Linked Open Services (LOS) [16]. LOS descriptions also use basic graph patterns for defining service inputs and outputs. In contrast to our work, LOS consume Linked Data instead of literals and URIs representing the required inputs. The LOS approach has the advantage that the relation between the inputs is made explicit in the service call. With LIDS the relations are only implicitly given by the input description. This drawback is in our view compensated by the advantage that LIDS service calls are directly linkable from within Linked Data, as service inputs can be encoded in a URI.

Other related work to integrating data comes from the database community, specifically information integration. Mediator systems (e.g., Information Manifold [17]) are able to answer queries over heterogeneous data sources, including services on the Web. Information-providing data services were explicitly treated, e.g., in [18, 19]. For an extensive overview of query answering in information integration systems, we refer the reader to [8]. All these works have in common that they answer queries using services, but do not provide methods to expose services with a standardised interface and link-able interfaces. Thus information integration is only done at the time of query answering, which is in contrast to our proposed approach that allows data sets to be directly interlinked, independent of a query processor.

8 Conclusions

A large portion of data on the Web is attainable through a large number of data services with a variety of interfaces that require procedural code for the integration of different data sources. We presented a general method for exposing data services as Linked Data, which enables the integration of different data sources without specialised code. Our method includes an interface convention that allows service inputs to be given as URIs and thus linked from other Linked Data sources. By exposing URIs for service inputs in addition to service outputs, the model neatly integrates with existing data, can handle multiple outputs for one input and makes the relation between input and output data explicit.

Furthermore, we proposed a lightweight description formalism and showed how it can be used for automatically interlinking Linked Data Services with appropriate data sets. We showed how the descriptions can be instantiated in SPARQL. We applied our method to create LIDS for existing real-world service, thus contributing new data to the Web. The approach was evaluated for performance and effectiveness in an experiment in which we interlinked the Billion Triple Challenge (BTC) 2010 data set with the GeoNames LIDS wrapper. We showed that the algorithm scales even to this very large data set and produces large numbers (around 2.5 million) of new links between entities. A possible av-

enue for future work would be to integrate fuzzy matching algorithms, similar to [10], in case the input to a web service is ambiguous, e.g., for services which take keywords as input.

We further plan future work in three main areas:

- improve tool support, so that Semantic Web developers can easily adopt the LIDS method for their applications and services;
- develop approaches for integrating LIDS into SPARQL query processing;
- integrate provenance information and usage policies in the service descriptions, in order to ensure legal compliance and traceability of integrated data sets.

Acknowledgements

The authors acknowledge the support of the European Community's Seventh Framework Programme FP7/2007-2013 (PlanetData, Grant 257641) and of the Deutsche Forschungs Gemeinschaft (Information Management and Engineering Graduate School, GRK 895).

References

1. Speiser, S., Harth, A.: Taking the LIDS off Data Silos. In: Triplification Challenge at I-SEMANTICS. (2010)
2. Speiser, S., Harth, A.: Towards Linked Data Services. In: The Semantic Web - Posters and Demonstrations (ISWC). (2010)
3. Berners-Lee, T.: Linked Data. Design Issues (2009) `http://www.w3.org/DesignIssues/LinkedData`.
4. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Rec. (Feb 2004) `http://www.w3.org/TR/rdf-concepts/`.
5. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. **2** (May 2002) 115–150
6. Bizer, C., Cyganiak, R., Gauss, T.: The RDF Book Mashup: From Web APIs to a Web of Data. In: Workshop on Scripting for the Semantic Web. (2007)
7. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: International Conference on Very Large Data Bases (VLDB). (2001) 129–138
8. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal **10** (2001) 270 – 294
9. Troncy, R., Fialho, A., Hardman, L., Saathoff, C.: Experiencing events through user-generated media. In: First International Workshop on Consuming Linked Data (COLD2010). (2010)
10. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: The Semantic Web (ISWC). (2009)
11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services (2004) `http://www.w3.org/Submission/OWL-S/`.

12. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Applied Ontology **1**(1) (2005) 77–106
13. Iqbal, K., Sbodio, M.L., Peristeras, V., Giuliani, G.: Semantic Service Discovery using SAWSDL and SPARQL. In: International Conference on Semantics, Knowledge and Grid (SKG). (2008)
14. Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., Stevens, R.: Deciding Semantic Matching of Stateless Services. AAAI Conference on Artificial Intelligence (AAAI) (2006)
15. Zhao, W.F., Chen, J.L.: Toward Automatic Discovery and Invocation of Information-Providing Web Services. In: Asian Semantic Web Conference (ASWC). (2006)
16. Norton, B., Krummenacher, R.: Consuming dynamic linked data. In: First International Workshop on Consuming Linked Data (COLD2010). (2010)
17. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: International Conference on Very Large Data Bases (VLDB). (1996)
18. Thakkar, S., Ambite, J.L., Knoblock, C.A.: A Data Integration Approach to Automatically Composing and Optimizing Web Services. In: Workshop on Planning and Scheduling for Web and Grid Services. (2004)
19. Barhamgi, M., Champin, P.A., Benslimane, D.: A Framework for Web Services-Based Query Rewriting and Resolution in Loosely Coupled Information Systems (2007)

[P3] Andreas Harth and Sebastian Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*. 2012.

Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

On Completeness Classes for Query Evaluation on Linked Data

Andreas Harth and **Sebastian Speiser**
Institute AIFB
Karlsruhe Institute of Technology (KIT)
76128 Karlsruhe, Germany

Abstract

The advent of the Web of Data kindled interest in link-traversal (or lookup-based) query processing methods, with which queries are answered via dereferencing a potentially large number of small, interlinked sources. While several algorithms for query evaluation have been proposed, there exists no notion of completeness for results of so-evaluated queries. In this paper, we motivate the need for clearly-defined completeness classes and present several notions of completeness for queries over Linked Data, based on the idea of authoritativeness of sources, and show the relation between the different completeness classes.

1 Introduction

A tenet in work on query evaluation and reasoning on the Semantic Web is the open world assumption (OWA): given the size and decentralised nature of the web, it is impossible to achieve complete results. Thus, an answer to a query or reasoning task is therefore always a subset of all possible answers. To what degree that subset is complete is left open.

In this paper we define more fine-grained completeness classes for query answers. We do so in the context of Linked Data, a set of principles detailing how to publish graph-structured data on the web. Recently developed query evaluation algorithms traverse the Web of Data and at the same time record answers to a query (Hartig, Bizer, and Freytag 2009; Harth et al. 2010; Ladwig and Tran 2010; Haase, Mathäß, and Ziller 2010; Umbrich, Hogan, and Polleres 2011). These algorithms, however, lack a clear specification of result completeness.

Thus, we present several completeness classes that rigorously define which sources may contribute to an answer to Linked Data queries. Doing so has a number of benefits; with a clear specification of complete answers:

- users know what to expect from a query evaluation algorithm;
- different algorithms become comparable;
- algorithms can have crisp termination criteria;
- developers can devise optimised algorithms that exclude irrelevant sources;
- systems can implement operations that rely on checking for the absence of results, such as negation-as-failure; and
- certain statements can be restricted to trustworthy sources.

Our specific contributions are:

- We extend and generalise the idea of authoritative sources from (Hogan, Harth, and Polleres 2009); based on authority, we define the notion of completeness for sources.
- We define three completeness classes for triple patterns and conjunctive queries: one that considers the entire web, one that considers documents in the surrounding of sources derived from the query and one that considers documents based on the query execution.
- We show how the completeness classes related to each other.

Please note that our results apply to both web and intranet environments, as long as data providers follow Linked Data principles. Our results also apply to Dataspaces (Franklin, Halevy, and Maier 2005) without central registries.

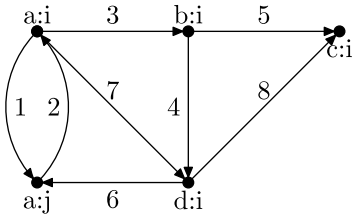
The remainder of the paper is organised as follows: Section 2 provides an example. Section 3 introduces necessary notation and definitions. Section 4 presents the idea of authoritative documents. Section 5 explains how query parts can be answered completely, while Section 6 considers entire queries under three completeness types. Section 7 explains the relation between the completeness classes. Section 8 presents related work, and Section 9 concludes.

2 Example

We begin with an example of an RDF graph and a query over that graph.

Example 1. *Figure 1 shows an example RDF graph. We use labels* `a:i`, `a:j`, `b:i`, `c:i`, `d:i`, `p:i` *to denote resources, and numbers* $1 \ldots 8$ *to denote triples. Now, assume the query* Q_{ex} *depicted in Figure 2. The overall goal is to find bindings* μ *to the variables in the query.*

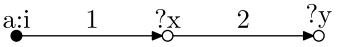
A system with access to the entire graph in Figure 1 could evaluate the query using standard query processing techniques. However, on the Linked Data web, the graph is distributed across multiple sources in form of web-accessible RDF files (henceforth called documents).



No	Triple
1	a:i p:i a:j .
2	a:j p:i a:i .
3	a:i p:i b:i .
4	b:i p:i d:i .
5	b:i p:i c:i .
6	d:i p:i a:j .
7	a:i p:i d:i .
8	d:i p:i c:i .

Document	Triple
a	1, 2, 3
b	4
c	5
d	6, 7
e	8
p	-

Figure 1: An example RDF graph with six IRIs and eight triples. Numbers denote triples.



No	Triple pattern
1	a:i p:i ?x .
2	?x p:i ?y .

Figure 2: An acyclic query consisting of two triple patterns.

Assume a, b, c, d, e, p are documents; the right table in Figure 1 lists the six documents and the triples they contain. Please note that the assignment is rather arbitrary and can differ, as maintainers of documents are free to decide which triples they host. One thing we can assume, though, is that identifiers are associated with documents (as mandated by the Linked Data principles (Berners-Lee 2006)). Thus, we can assume that we get some triples with identifer a:i when looking up the corresponding document a (and similarly, a:j for a and b:i for b, c:i for c).

Table 1: Bindings for variables in Q_{ex}, including which triples and documents contributed to bindings.

μ	μ(?x)	μ(?y)	Triple	Document
μ_1	a:j	a:i	1, 2	a
μ_2	b:i	d:i	3, 4	a , b
μ_3	d:i	a:j	7, 6	d

Now, to answer the query, we perform a lookup on a, which results in triples 1 - 3 from which we can derive bindings a:j and b:i for ?x, and a:i for ?y. Next, we perform a lookup on b which returns triple 4, from which we can derive d:i for ?y. We also perform a lookup on d which returns triple 7 and 6, from which we can derive d:i for ?x and a:j for ?y. As a result, we arrive at the bindings as depicted in Table 1. Please note that bindings $\{?x \mapsto b:i, ?y \mapsto c:i\}$ (via sources a and c) and bindings $\{?x \mapsto a:i, ?y \mapsto c:i\}$ (via sources d and e) cannot be reached via link traversal.

The example illustrates a couple of issues: first, a link-traversal algorithm cannot discover documents which are not referenced in any already known document. Second, assuming a larger graph, the link traversal process could actually go on for a long time, as more and more new documents are discovered and accessed. In the rest of the paper we show how to decide which subset of documents should be accessed to derive answers to queries.

3 Preliminaries

We introduce basic notation to clarify our understanding of RDF, Linked Data and queries. We stay close to similar definitions as found in (Pérez, Arenas, and Gutierrez 2009; Umbrich, Hogan, and Polleres 2011).

Definition 1 (RDF Terms, Triple, Graph). *The set of RDF terms consists of the set of IRIs $\mathcal{I}$, the set of blank nodes $\mathcal{B}$ and the set of literals $\mathcal{L}$. A triple $(s, p, o) \in \mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object. We denote by $\mathsf{s}(t)$ the subject, $\mathsf{p}(t)$ the predicate and $\mathsf{o}(t)$ the object of a triple t. We denote by $\mathsf{iris}(t)$ all IRIs from a triple t, and by $\mathsf{terms}(t)$ all RDF terms. A set of triples is called RDF graph; $\mathcal{G} = 2^{\mathcal{T}}$ is the set of all graphs.*

Next, we define ways for accessing RDF graphs published on the web as Linked Data. A key characteristic of Linked Data is the correspondence between an identifier and a source; i.e., the name for a thing (non-information resource) is associated with the document where one can find related information (information resource).

Definition 2 (Information Resource, Lookup). *Let $\mathcal{I}_{\mathcal{I}} \subseteq \mathcal{I}$ be the set of all information resources. The set of all non-information resources is defined as $\mathcal{I}_{\mathcal{N}} = \mathcal{I} \setminus \mathcal{I}_{\mathcal{I}}$. The function $\mathsf{deref} : \mathcal{I}_{\mathcal{I}} \mapsto \mathcal{G}$ models a Linked Data lookup and returns the graph represented in a document, or the empty set if none found, e.g., if there is a timeout or the document returns non-RDF content.*

We use the terms information resource and document interchangably. To be able to model the assocation between non-information Resources and information resources we introduce the concept of correspondence.

Definition 3 (Correspondence). *The function $\mathsf{co} : \mathcal{I} \mapsto \mathcal{I}_{\mathcal{I}}$ associates to a resource its information resource. For inputs from $\mathcal{I}_{\mathcal{I}}$, co behaves as the identity function.*

Determining the kind of an IRI is not always possible from the outset; a HTTP lookup clarifies the kind of IRI. We define a high-level function which provide abstractions on low-level functionality pertaining to protocol-level issues. Thus, in co we abstract away the following cases:

1. remove the local identifier from an IRI (i.e., strip everything after the # symbol);
2. dereference the IRI and follow redirects (HTTP status codes 30x);

3. dereference the IRI and parse the `Content-Location` header to yield the canonical name;
4. no-op: do nothing if the IRI is an information resource.

Options 1-3 may be called never or repreatedly, to ultimately arrive at 4. The co function may never return due to infitite redirects; in practice, one sets a limit on how often co can be applied.

Definition 4 (Variable, Triple Pattern). *Let $\mathcal{V}$ be a set of variables; variables bind to RDF terms from $\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. A triple $p \in (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{L} \cup \mathcal{V})$ is called triple pattern. We omit blank nodes from triple patterns for ease of exposition. $\mathcal{P}$ is the set of all triple patterns. We denote by $\mathsf{vars}(p)$ all variables from a triple pattern p.*

Definition 5 (Variable Binding). *Let $\mathcal{M}$ be the set of all partial functions $\mu\colon \mathcal{V} \mapsto \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. A function $\mu \in \mathcal{M}$ is called a variable binding.*

Definition 6 (Basic Graph Pattern (BGP)). *A BGP (or just query) is a set $Q \subset \mathcal{P}$. The set of all queries is $\mathcal{Q} = 2^{\mathcal{P}}$.*

BGP queries are important as they present a large subset of SPARQL. Previous work also focussed on such queries.

Definition 7 (Query Binding). *The bindings of a query $Q \in \mathcal{Q}$ on an RDF graph $G \in \mathcal{G}$ consisting of the triples available at a set $I \subset \mathcal{I}_\mathcal{I}$, denoted as $\mathsf{bindings}\colon \mathcal{Q} \times 2^{\mathcal{I}_\mathcal{I}} \mapsto 2^{\mathcal{M}}$, is the set of minimal variable bindings which map Q to a subgraph of G: $\mathsf{bindings}(Q, I) = \{\mu \in \mathcal{M} \mid \mathsf{dom}(\mu) = \mathsf{vars}(Q) \wedge \forall p \in Q.\mu(p) \in \cup_{u \in I} \mathsf{deref}(u)\}$.*

4 Authoritative Documents

We introduce the notion of authoritative document for an identifier, that is, we define which information resource can talk authoritatively about a specific identifier. In other words, we restrict the documents which can make statements containing certain identifiers. Our notion is an extension and generalisation of the idea of authoritative source from (Hogan, Harth, and Polleres 2009).

The notion of authoritativeness is important on the web, which consists of a motley collection of data sources, some of which may provide questionable information. Also, we use authoritativeness to specify which information resources are necessary to have complete information about an identifier.

Definition 8 (Authoritative Document). *Document u talks with authority about a triple t if there is a correspondence between u and any identifier from t, i.e., $\mathsf{co}(s(t)) = u$, $co(p(t)) = u$ or $co(o(t)) = u$. We call a document u to be subject-authoritative for t if $\mathsf{co}(s(t)) = u$ (s-auth in short). Analogously, p-auth and o-auth relate a document to the identifier of a predicate or object.*

Example 2. *Consider the triples and documents from Figure 1. Document `a` talks with authority about triples 1-3, namely s-auth for 1-3 and o-auth for triples 1 and 2. Document `e` contains triple 8 using identifiers (`d:i`, `p:i`, `c:i`) without authority, as there is no connection in* co *between any of the identifiers and `e`.*

Definition 9 (Authority Types). *We can have atomtic authority types s, p, or o denoting whether a triple has been stated with authority regarding its subject, predicate or object. We can combine atomic authority types using conjunction and disjunction to arrive at the set of possible authority types $\mathcal{A} = \{\bot, s, p, o, s \vee p, s \vee o, p \vee o, s \vee p \vee o, s \wedge p, s \wedge o, p \wedge o, s \wedge p \wedge o\}$. Note that $\bot$ denotes no authority.*

Example 3. *In the following, we explain two examplaric authority types:*

- *A triple t is stated $s \wedge o$-auth, if both $t \in \mathsf{deref}(\mathsf{co}(s(t)))$ and $t \in \mathsf{deref}(\mathsf{co}(o(t)))$.*
- *A triple t is stated $s \vee p \vee o$-auth, if $t \in \mathsf{deref}(\mathsf{co}(s(t)))$ or $t \in \mathsf{deref}(\mathsf{co}(p(t)))$ or $t \in \mathsf{deref}(\mathsf{co}(o(t)))$.*

Based on the notion of authority types we introduce a modified deref function, the derefa function, which only selects triples that satisfy specified authority types.

Definition 10 (Authoritative Lookup). *The function $\mathsf{derefa}\colon \mathcal{I}_\mathcal{I} \times \mathcal{A} \mapsto \mathcal{G}$ models a Linked Data lookup and returns the graph represented in an information resource, while applying the specified authority types, i.e., filtering the triples which do not adhere to the authority criteria. Please note that* derefa *might perform additional lookups if those are required for clarifying the authoritativeness of a triple.*

Example 4. *The function $\mathsf{derefa}(b, s \wedge o)$ involves $\mathsf{deref}(b)$, yielding the triple (`b:i` , `p:i` , `d:i`) and subsequently requiring also a $\mathsf{deref}(\mathsf{co}(d\texttt{:}i))$ to verify that the triple also occurs in d.*

In case $a = \bot$ the results for deref and derefa coincide.

The different authority types specify the documents that can contribute certain triples to query results, thus paving the way towards defining completeness.

5 Authoritative Documents for Triple Patterns

We now show which documents are relevant to a triple pattern p under a specified authority type a. If we assure that these relevant documents are dereferenced with the derefa function, we can state that p has been completely answered under a. Based on complete answers to single triple patterns, we define complete answers to Basic Graph Pattern queries in Section 6.

Consider a triple pattern p for which we want to get bindings. In Linked Data query evaluation, the query processor has to dereference (lookup) IRIs which yields data, which in turn is matched with the triple pattern to ultimately yield bindings.

Thus, to get all possible bindings on the web, we would need to get all $\mathcal{I}_\mathcal{I}$ and match the resulting graphs to the triple pattern p. However, based on the notion of authoritative source, we can answer a triple pattern p completely, given a defined authority type.

Example 5. *Consider the triple pattern $p_1 = (\texttt{a:i}, \texttt{p:i}, \texttt{?x})$. If we restrict the answers to be derived from s-auth triples, we are sure to get all those triple if we perform a lookup on `a:i`, that is, $\mathsf{derefa}(\mathsf{co}(\texttt{a:i})), s)$. Thus, we have answered p_1 completely under s-auth assumption.*

We now use the definition of $\mathcal{A}$ to derive, given a triple pattern and authority specification, the subset of $\mathcal{I}$ we have to dereference to find the complete set of bindings for the pattern.

Definition 11 (Completely Sufficient Documents). *We define* $\mathsf{csuff}\colon \mathcal{P} \times \mathcal{A} \mapsto 2^{2^{\mathcal{I}\times\mathcal{V}}}$*, which, given a pattern and an authority type, returns a set of alternative documents sets, each of which is sufficient to completely answer the triple pattern.*

The use of variables becomes clear in Section 6.

$$\mathsf{csuff}(t,a) = \begin{cases} \{\{\mathcal{I}_\mathcal{I}\}, & \text{if } a = \bot \\ \{\{\mathsf{s}(t)\}\}, & \text{if } a = s \\ \{\{\mathsf{p}(t)\}\}, & \text{if } a = p \\ \{\{\mathsf{o}(t)\}\}, & \text{if } a = o \\ \{\{\mathsf{s}(t)\},\{\mathsf{p}(t)\}\}, & \text{if } a = s \wedge p \\ \{\{\mathsf{s}(t)\},\{\mathsf{o}(t)\}\}, & \text{if } a = s \wedge o \\ \{\{\mathsf{p}(t)\},\{\mathsf{o}(t)\}\}, & \text{if } a = p \wedge o \\ \{\{\mathsf{s}(t)\},\{\mathsf{p}(t)\},\{p(o)\}\}, & \text{if } a = s \wedge p \wedge o \\ \{\{\mathsf{s}(t),\mathsf{p}(t)\}\}, & \text{if } a = s \vee p \\ \{\{\mathsf{s}(t),\mathsf{o}(t)\}\}, & \text{if } a = s \vee o \\ \{\{\mathsf{p}(t),\mathsf{o}(t)\}\}, & \text{if } a = p \vee o \\ \{\{\mathsf{s}(t),\mathsf{p}(t),\mathsf{o}(t)\}\}, & \text{if } a = s \vee p \vee o \end{cases}$$

Note that when no authority type is given ($a = \bot$), we would need to retrieve the set of all documents to arrive at complete answers. There can be several alternatives that are sufficient for completely answering a pattern (see $s \wedge o$ authority), and each alternative can require more than one position (see $s \vee o$ authority).

If we know that a triple t exists in the documents $\mathcal{I}_\mathcal{I}$ under an authority type a, we can infer that t exists in the corresponding document of one IRI of each alternatively sufficient IRI set (denoted as L for aLternative):

$$\begin{aligned} & t \in \mathsf{derefa}(\mathcal{I}_\mathcal{I}, a) \\ \rightarrow & \forall L \in \mathsf{csuff}(t,a).\exists l \in L.t \in \mathsf{derefa}(\mathsf{co}(l), a). \end{aligned}$$

The fact that the triple must be contained in all alternatives may not sound intuitive at first, but every alternative is sufficient to determine whether the triple exists.

Example 6. *Consider the triple pattern* $p_1 = (\texttt{a:i}, \texttt{p:i}, \texttt{?x})$*. We illustrate the complete answers for* p_1 *under different authority types:*

- *s-auth:* $\mathsf{csuff}(p_1, s) = \{\{\texttt{a:i}\}\}$*, so there is only one alternative for completely answering* p_1 *by finding all bindings* $\mu \in \mathcal{M}$*, such that* $\mu(p_1) \in \mathsf{derefa}(\texttt{a:i}, s)$*, which would result in the bindings* $\mu_1 = \{\texttt{?x} \mapsto \texttt{a:i}\}$*, and* $\mu_2 = \{\texttt{?x} \mapsto \texttt{b:i}\}$*.*
- $s \wedge p$*-auth:* $\mathsf{csuff}(p_1, s \wedge p) = \{\{\texttt{a:i}\},\{\texttt{p:i}\}\}$*, so it would be sufficient to retrieve either the graph* $\mathsf{derefa}(\texttt{a:i}, s \wedge p)$ *or the graph* $\mathsf{derefa}(\texttt{p:i}, s \wedge p)$ *to find all bindings for* p_1*. However, both graphs are empty, as there is no triple in* $\mathsf{co}(\texttt{p:i}) = \texttt{p}$ *and thus none of the triples in* $\mathsf{co}(\texttt{a:i}) = \texttt{a}$ *is "confirmed", as required by* $s \wedge p$ *authority. Please note that the invocation of* derefa *may involve additional lookups to ensure that triples adhere to a given authority type. These additional lookups only invalidate existing results but never contribute new ones.*
- $s \vee o$*-auth:* $\mathsf{csuff}(p_1, s \vee o) = \{\{\texttt{a:i}, \texttt{?x}\}\}$*, so we cannot answer* p_1 *completely, because there is only one alternative, which would require a binding for* ?x*.*

One complication arises when all alternatives returned by csuff contain variables instead of IRIs. In this case, the pattern cannot be completely answered under the authority scheme. However, if we have conjunctions of several triple patterns, another pattern may be used to find complete bindings for the variables in a sufficient alternative, thus making the conjunction completely answerable. We define completeness for such conjunctions in the next section.

6 Completeness of Basic Graph Patterns

In the following, we address the problem of answering queries consisting of several patterns (so-called Basic Graph Patterns). A query Q consisting of several patterns $tp_0, tp_1, \ldots tp_n$ can be completely answered if the corresponding required positions of a triple pattern are bound either by a constant or by a variable in another completely answerable pattern in the query. We thus define a mapping for assigning a required authority to every pattern in a query.

Definition 12 (Authority Mapping). *We define a mapping* $\alpha\colon Q \mapsto \mathcal{A}$ *that assigns triple patterns in* Q *to different authority types. The set of all such mappings is denoted as* $\mathcal{AU}$*.*

Definition 13 (Authoritative Query Bindings). *We extend the* $\mathsf{bindings}\colon \mathcal{Q} \times 2^\mathcal{I} \mapsto 2^\mathcal{M}$ *function to return only bindings satisfying an authority mapping* α*:* $\mathsf{bindings}^\alpha(Q, I) = \{\mu \in \mathcal{M} \mid \mathsf{dom}(\mu) = \mathsf{vars}(Q) \wedge \forall p \in Q.\mu(p) \in \cup_{u\in I}\mathsf{derefa}(u, \alpha(p))\}$.

We define completeness via a set s of documents that have to be retrieved to completely answer a query Q, i.e., a set s is complete for an authority mapping α, if $\mathsf{bindings}^\alpha(Q, s)$ contains all desired query results. A natural requirement for such a set s of documents is that it holds the same results for Q as the entire Linked Data web, i.e. $\mathsf{bindings}^\alpha(Q, s) = \mathsf{bindings}^\alpha(Q, \mathcal{I}_\mathcal{I})$.

As it is infeasible to materialise the entire Linked Data web, i.e., $\mathsf{deref}(\mathcal{I}_\mathcal{I})$, and thus instead we are searching for a subset $s \subset \mathcal{I}_\mathcal{I}$, where $|s| \ll |\mathcal{I}_\mathcal{I}|$, which can be accessed at query time and so that $\mathsf{deref}(s)$ contains sufficient information to answer the query Q.

Thus, we define that a set s of documents is complete for query Q given an authority mapping α, if $\mathsf{complete}(Q, \alpha) \subseteq s$, where complete is one of the different completeness classes introduced in the following:

- web-complete $\mathsf{wc}\colon \mathcal{Q} \times \mathcal{AU} \mapsto 2^{\mathcal{I}_\mathcal{I}}$ which is mainly of theoretical interest when considering the web, but possibly applicable to controlled environments such as intranets;
- seed-complete $\mathsf{sc}\colon \mathcal{Q} \times \mathcal{AU} \mapsto 2^{\mathcal{I}_\mathcal{I}}$ which is practical and pragmatic solution, if no authority restrictions are given;

- query-reachable-complete $\mathsf{qrc}\colon \mathcal{Q} \times \mathcal{AU} \mapsto 2^{\mathcal{I}_\mathcal{I}}$ which defines complete results under given authority types for a certain class of queries.

We now formally define the three different completeness classes and then discuss the relationships between the different notions in Section 7.

6.1 Web-complete Set

The web-complete set gives the results of the query, when it is evaluated over the whole Linked Data web, i.e. $\mathcal{I}_\mathcal{I}$. However it is sufficient to evaluate over every document that helps to produce a result binding, (could also be a duplicate of a binding that can be produced without it). Without authority restrictions, every document can contain arbitrary triples, thus there is no other way of determining the set than accessing every $u \in \mathcal{I}_\mathcal{I}$ or having some form of index structure, which has accessed every such u before.

Definition 14. *With authority restriction, we define web-complete as the set of documents that contain a triple which is part of a result when evaluating Q over $\mathcal{I}_\mathcal{I}$.*

$$\mathsf{wc}(Q,\alpha) = \{u \in \mathcal{I}_\mathcal{I} \mid \exists \mu \in \mathsf{bindings}^\alpha(Q,\mathcal{I}_\mathcal{I}). \exists p \in Q.\ \mu(p) \in \mathsf{derefa}(u,\alpha(p))\}.$$

Example 7. *Considering our example of query $\mathrm{Q_{ex}}$ and assuming two authority mappings α_1 and α_2 we get:*

- *Let $\alpha_1(p) = \bot$, for $p \in \mathrm{Q_{ex}}$:* $\mathsf{wc}(\mathrm{Q_{ex}},\alpha_1) = \{a,b,c,d,e\}$.
- *Let $\alpha_2(p) = s$, for $p \in \mathrm{Q_{ex}}$:* $\mathsf{wc}(\mathrm{Q_{ex}},\alpha_2) = \{a,b\}$.

6.2 Seed-complete Set

The seed-complete set consists of all documents that can be reached via following triple paths of maximum length of the query beginning from triples in the documents identified by the IRIs in the query. The intuition is a traversal of $\mathcal{I}_\mathcal{I}$ to get the documents that are up to n hops away.

In the size-restricted seed-complete set, we fix n to the query size $|Q|$. In an alternative, length-restricted seed-complete set (which we leave open for future work), we can fix n to the depth of the query, i.e., the length of longest path in the query, starting from a constant.

As there can be several different IRIs in the query and from each IRI there can start several paths of triples, we possibly end up with forests, consisting of several trees starting in different triples.

Definition 15 (Forest). *A triple forest grounded in a set of seed IRIs is a list of triples, where each triple is either in the seed IRIs, or in the corresponding document of a resource occuring in a previous triple in the list. The function* $\mathsf{forests}\colon 2^{\mathcal{I}_\mathcal{I}} \times \mathbb{N} \mapsto 2^{\mathcal{T}^*}$ *returns all forests of triples of size up to n, starting with the triples in the seed IRIs $s_0 = \mathsf{co}(\mathsf{iris}(Q))$:*

$$\mathsf{forests}(s_0,n) = \bigcup_{j\in[1..n]} \{(t_1,\ldots,t_j) \in \mathcal{T}^j \mid \forall i \in [1..j].t_i \in \cup_{u\in s_0}\mathsf{deref}(u) \vee \exists k \in [1..i-1].t_i \in \cup_{u\in\mathsf{iris}(t_k)}\mathsf{deref}(u)\}$$

Definition 16. *We define the seed completeness set to contain all documents corresponding to IRIs in the forests grounded in the query's IRIs:*

$$\mathsf{sc}(Q,\alpha) = \mathsf{co}(\mathsf{iris}(\mathsf{forests}(\mathsf{iris}(Q),|Q|))).$$

Example 8. *Considering our example of query $\mathrm{Q_{ex}}$, we get for $\mathsf{iris}(\mathrm{Q_{ex}}) = \{\texttt{a:i}, \texttt{p:i}\}$, and $|\mathrm{Q_{ex}}| = 2$:*

$$\begin{aligned}\mathsf{forests}(\mathsf{iris}(\mathrm{Q_{ex}}),|\mathrm{Q_{ex}}|) = \{&(t_1),(t_1,t_1),(t_1,t_2),(t_1,t_3),\\ &(t_2),(t_2,t_1),(t_2,t_2),(t_2,t_3),\\ &(t_3),(t_3,t_1),(t_3,t_2),(t_3,t_3),\\ &(t_3,t_4)\}.\end{aligned}$$
$$\mathsf{sc}(\mathrm{Q_{ex}},\alpha) = \{a,p,b,d\},$$

where t_i stands for triple number i from the runnnig example (see Section 2).

6.3 Query-reachable-complete Set

We first define the notion of completely answerable queries for a given authority mapping α. Then, we specify the set of documents required to answer such a query completely in the sense of obtaining the same results as if the query would be evaluated over the web-complete set. The equivalence of the result sets is shown in Section 7.

Definition 17 (Completely-answerable Query). *A query is completely answerable if the triple patterns can be brought into an order, such that for each triple pattern p, there exists a set of RDF terms sufficient to completely answer p, where each term is either an IRI or a variable occuring in a previous pattern. The predicate caq^α defines the completely answerable property of a query under an authority mapping α:*

$$\begin{aligned}&\mathsf{caq}^\alpha(Q) \leftrightarrow\\ &(Q = \{p\} \wedge \exists L \in \mathsf{csuff}(p,\alpha(p)).\forall l \in L.l \in \mathcal{I}_\mathcal{N}) \vee\\ &(|Q| > 1 \wedge \exists Q_n, Q_1.Q_n \cup Q_1 = Q \wedge Q_n \cap Q_1 = \emptyset \wedge\\ &\qquad \mathsf{caq}^\alpha(Q_n) \wedge Q_1 = \{p\} \wedge\\ &\qquad \exists L \in \mathsf{csuff}(p,\alpha(p)).\forall l \in L.l \in \mathcal{I} \vee l \in \mathsf{vars}(Q_n)).\end{aligned}$$

In other (recursive) words: a query Q is completely answerable if either Q is of size 1 and there exists a set of sufficient terms which are all IRIs in Q, or one can remove a pattern p from the query, such that the resulting query Q_n is completely answerable, and p has a set of required terms which are either IRIs or variables bound by query Q_n.

An IRI must be in the query-reachable-complete set if it occurs in a forest, starting in the IRIs of the query, which is a result for a completely answerable subquery of the original query.

Definition 18 (Completely Answerable Subqueries). *The function* $\mathsf{csq}^\alpha\colon \mathcal{Q} \mapsto 2^\mathcal{Q}$ *returns all completely-answerable subqueries of a query:*

$$\mathsf{csq}^\alpha(Q) = \{Q' \subseteq Q \mid \mathsf{caq}^\alpha(Q')\}.$$

The forests, which are results for a completely answerable subquery of Q are defined by the function $\mathsf{qforests}^\alpha$ as

a subset of all forests starting in the IRIs contained in the query.

$$\mathsf{qforests}^\alpha(Q) = \{F \in \mathsf{forests}(\mathsf{co}(\mathsf{iris}(Q)), |Q|) \mid \exists Q' \in \mathsf{csq}^\alpha(Q) \wedge \exists \mu \in \mathcal{M}.\mu(Q') = F\}$$

Definition 19. *We define the query-reachable-complete set, to contain all documents corresponding to IRIs in the forests that produce bindings for a completely answerable subquery of Q:*

$$\mathsf{qrc}(Q, \alpha) = \mathsf{co}(\mathsf{iris}(\mathsf{qforests}^\alpha(Q))).$$

Example 9. *Considering our example of query $\mathrm{Q_{ex}}$, we get for α, where $\alpha(p) = s$, for all $p \in \mathrm{Q_{ex}}$:*

$$\begin{aligned}\mathsf{csq}^\alpha(\mathrm{Q_{ex}}) &= \{\mathrm{Q_{ex}}, \{(\texttt{a:i}, \texttt{p:i}, \texttt{?x})\}\}\\ \mathsf{qforests}^\alpha(\mathrm{Q_{ex}}) &= \{(t_1), (t_1, t_2), (t_3), (t_3, t_4)\}\\ \mathsf{qrc}(\mathrm{Q_{ex}}, \alpha) &= \{\mathtt{a}, \mathtt{p}, \mathtt{b}, \mathtt{d}\},\end{aligned}$$

where t_i stands for triple number i from the runnnig example (see Section 2). We can see that $\mathsf{wc}(\mathrm{Q_{ex}}, \alpha) \subset \mathsf{qrc}(\mathrm{Q_{ex}}, \alpha)$, meaning that the query reachable set produces all bindings available in the web under the authority mapping α. In Section 7 we show this in general for all completely answerable queries.

7 Relations Between Completeness Classes

In the following, we show the relation between the different completeness classes.

Theorem 1. *QRC results are a subset of (size-restriced) SC results:* $\mathsf{qrc}(Q, \alpha) \subseteq \mathsf{sc}(Q, \alpha)$.

Proof. Theorem 1 is obvious from the definition, as $\mathsf{qforests}^\alpha(Q) \subseteq \mathsf{forests}(\mathsf{co}(\mathsf{iris}(Q), |Q|))$. □

Theorem 2. *SC query results are a subset of WC results:* $\mathsf{bindings}^\alpha(Q, \mathsf{sc}(Q, \alpha)) \subseteq \mathsf{bindings}^\alpha(Q, \mathsf{wc}(Q, \alpha))$.

Proof. Theorem 2 is obvious from the definition, as web complete is defined to contain all bindings, and thus seed complete can not contain more bindings. □

Theorem 3. *For a query Q that is completely answerable under an authority mapping α, the bindings for query reachable complete and web complete coincide:* $\mathsf{bindings}^\alpha(Q, \mathsf{qrc}(Q, \alpha)) = \mathsf{bindings}^\alpha(Q, \mathsf{wc}(Q, \alpha))$.

Proof. We prove the equivalence of the sets, by showing their mutual containment:
(1) $\mathsf{bindings}^\alpha(Q, \mathsf{qrc}(Q, \alpha)) \subseteq \mathsf{bindings}^\alpha(Q, \mathsf{wc}(Q, \alpha))$ follows from the definition, that web completeness means that every (in this case α-authoritative) result is found.
(2) $\mathsf{bindings}^\alpha(Q, \mathsf{wc}(Q, \alpha)) \subseteq \mathsf{bindings}^\alpha(Q, \mathsf{qrc}(Q, \alpha))$ is shown by induction on the query size. As bindings is monotonic over the set of documents, we reduce this case to showing that $\mathsf{wc}(Q, \alpha) \subseteq \mathsf{qrc}(Q, \alpha)$, if $\mathsf{caq}^\alpha(Q)$.
Induction start for a query Q of size 1:
from $\mathsf{caq}^\alpha(Q) \wedge |Q| = 1 \wedge u \in \mathsf{wc}(Q, \alpha)$ follows that there exists a binding μ for Q over $\mathcal{I}_\mathcal{I}$, which maps the single triple pattern $p \in Q$ to a triple from u: $\exists \mu \in \mathsf{bindings}^\alpha(Q, \mathcal{I}_\mathcal{I}).\mu(p) \in \mathsf{derefa}(u, \alpha(p))$. This implies that $u \in \mathsf{co}(\mathsf{iris}(\mu(p)))$, for $\alpha(p) \neq \bot$, which is ruled out by the definition of caq. Furthermore, we know that $(\mu(p)) \in \mathsf{forests}(\mathsf{co}(\mathsf{iris}(Q)), 1)$, as $\mu(p)$ is a query answer to $Q = \{p\}$, and p must be completely answerable, given that $\mathsf{caq}^\alpha(Q)$, it follows, that $(\mu(p)) \in \mathsf{qforests}^\alpha(Q)$ and thus: $u \in \mathsf{qrc}(Q, \alpha)$.
We form the induction hypothesis:

$$\mathsf{caq}^\alpha(Q) \wedge u \in \mathsf{wc}(Q, \alpha) \wedge |Q| = n \rightarrow u \in \mathsf{qrc}(Q, \alpha).$$

The inductive step: given $\mathsf{caq}^\alpha(Q) \wedge u \in \mathsf{wc}(Q, \alpha) \wedge |Q| = n + 1$, we can split Q into Q_n and Q_1, such that $Q = Q_n \cup Q_n \wedge Q_n \cap Q_1 = \emptyset \wedge |Q_1| = 1 \wedge \mathsf{caq}^\alpha(Q_n)$ (follows from $\mathsf{caq}^\alpha(Q)$). Accordingly our argument can be split into two cases:
Case (2.1): u is also in the web complete set of Q_n: $\mathsf{caq}^\alpha(Q_n) \wedge |Q_n| = n \wedge u \in \mathsf{wc}(Q_n, \alpha)$. We can use the induction hypothesis and conclude $u \in \mathsf{qrc}(Q_n, \alpha)$ and because the reachable completeness set is monotonic (a larger query still has the smaller query as a subquery), we conclude: $u \in \mathsf{qrc}(Q, \alpha)$.
Case (2.2): u is not in the web complete set of Q_n, thus it must be contributed by a variable binding or a constant in Q_1. Evaluating Q_1 has to be done only for the bindings of Q_n, other results that do not join with the results for Q_n cannot contribute an result for Q. As $\mathsf{caq}^\alpha(Q)$, we know that there exists a set of terms sufficient for completely answering Q_1, in which all terms are either constants or variables already occuring in Q_n. Therefore, we can reduce the case to considering only those $\mu(Q_1)$, where μ is a result binding for Q_n. Thus, $\mu(Q_1)$ is completely answerable, and we can use the induction start:

$$\mathsf{caq}^\alpha(\mu(Q_1)) \wedge |Q_1| = 1 \rightarrow u \in \mathsf{qrc}(Q_1, \alpha) \rightarrow u \in \mathsf{qrc}(Q, \alpha).$$

□

8 Related Work

Early work on queries over the web graph include (Mendelzon and Milo 1997) and (Abiteboul and Vianu 2000). (Hartig, Bizer, and Freytag 2009) introduced Linked Data query processing via link traversal. Subsequent work (Hartig, Bizer, and Freytag 2009; Harth et al. 2010; Ladwig and Tran 2010; Haase, Mathäß, and Ziller 2010; Umbrich, Hogan, and Polleres 2011) lack a rigorous specification of termination criteria (some use heuristics). (Fionda, Gutierrez, and Pirrò 2011) introduce a navigation language, which has different characteristics than our query language based on BGPs. (Hartig 2012) analyses the computability of SPARQL queries over Linked Data under different semantics. The notion of semantics in (Hartig 2012) roughly corresponds to our notion of completeness: the full-web semantics is similar to our web-completeness, whereas the reachability-based semantics can be considered as an abstract concept, while we provide two actual completeness classes. While (Hartig 2012) shows that in the general case

the full-web semantics is not computable, we show that under certain authority constraints it is possible to achieve results equivalent to web-completeness.

Several other papers propose definitions for semantics in distributed settings. The Local Relational Model (Serafini et al. 2003) uses model-theoretic means to specify the semantics of a federation of relational databases. Context OWL (Bouquet et al. 2003) considers description logic inference and connects models via bridge rules, whereas our definitions are on the RDF graph level with shared use of identifiers. Our definitions have an operational aspect and are tied to a view that assumes only local knowledge, in contrast to the global view taken by typical model-theoric approaches. Finally, (Polleres, Feier, and Harth 2006) use a form of a local closed world model to specify the semantics of queries with negation as failure under a modified OWA.

9 Conclusion

We have provided a general notion of authoritativeness and defined three completeness classes for Linked Data query evaluation. While the seed-complete class is straightforward to implement, the query-reachable-complete class requires less lookups while yielding a well-defined and useful set of results, based on a specified authority assignment for each query. However, implementing query-reachable is more intricate.

Future work includes extending the authority types with negation, thus allowing for a negation-as-failure semantics, an algorithm for enumerating possible authority assignments for queries, and an investigation of query result completeness when allowing sources with query capabilities.

Acknowledgements

We thank Sebastian Rudolph for insightful comments, and acknowledge the support of the European Commission's Seventh Framework Programme FP7/2007-2013 (PlanetData, Grant 257641).

References

Abiteboul, S., and Vianu, V. 2000. Queries and computation on the web. *Theoretical Computer Science* 239:231–255.

Berners-Lee, T. 2006. Linked Data. http://www.w3.org/DesignIssues/LinkedData.

Bouquet, P.; Giunchiglia, F.; van Harmelen, F.; Serafini, L.; and Stuckenschmidt, H. 2003. C-OWL: Contextualizing ontologies. In *Second International Semantic Web Conference*, number 2870 in Lecture Notes in Computer Science, 164–179. Springer.

Fionda, V.; Gutierrez, C.; and Pirrò, G. 2011. Semantic navigation on the web of data: Specification of routes, web fragments and actions. *CoRR* abs/1111.4316.

Franklin, M.; Halevy, A.; and Maier, D. 2005. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.* 34:27–33.

Haase, P.; Mathäß, T.; and Ziller, M. 2010. An evaluation of approaches to federated query processing over linked data. In *6th International Conference on Semantic Systems*, I-SEMANTICS 2010. ACM.

Harth, A.; Hose, K.; Karnstedt, M.; Polleres, A.; Sattler, K.-U.; and Umbrich, J. 2010. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th International Conference on World Wide Web*, 411–420. ACM.

Hartig, O.; Bizer, C.; and Freytag, J.-C. 2009. Executing sparql queries over the web of linked data. In *Eight International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, 293–309. Springer.

Hartig, O. 2012. Sparql for a web of linked data: Semantics and computability (extended version). *CoRR* abs/1203.1569.

Hogan, A.; Harth, A.; and Polleres, A. 2009. Scalable authoritative owl reasoning for the web. *International Journal on Semantic Web Information Systems* 5(2):49–90.

Ladwig, G., and Tran, T. 2010. Linked data query processing strategies. In *Ninth International Semantic Web Conference*, volume 6496 of *Lecture Notes in Computer Science*. Springer. 453–469.

Mendelzon, A. O., and Milo, T. 1997. Formal models of web queries. In *Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, 134–143. ACM.

Pérez, J.; Arenas, M.; and Gutierrez, C. 2009. Semantics and complexity of sparql. *ACM Transactions on Database Systems* 34:16:1–16:45.

Polleres, A.; Feier, C.; and Harth, A. 2006. Rules with contextually scoped negation. In *Third European Semantic Web Conference*, number 4011 in Lecture Notes in Computer Science, 332–347. Springer.

Serafini, L.; Giunchiglia, F.; Mylopoulos, J.; and Bernstein, P. 2003. Local relational model: a logical formalization of database coordination. In *Proceedings of the 4th International and Interdisciplinary Conference on Modeling and Using Context*, 286–299. Springer.

Umbrich, J.; Hogan, A.; and Polleres, A. 2011. Improving the recall of decentralised linked data querying through implicit knowledge. *CoRR* abs/1109.0181.

[P4] Anisa Rula, Matteo Palmonari, Andreas Harth, Steffen Stadtmüller, and Andrea Maurino. On the Diversity and Availability of Temporal Information in Linked Open Data. In *Proceedings of the 11th International Semantic Web Conference (ISWC)*, pages 492–507. 2012.

On the Diversity and Availability of Temporal Information in Linked Open Data

Anisa Rula[1], Matteo Palmonari[1], Andreas Harth[2], Steffen Stadtmüller[2], and Andrea Maurino[1]

[1] University of Milano-Bicocca
{rula,palmonari,maurino}@disco.unimib.it
[2] Karlsruhe Institute of Technology (KIT)
{harth,Steffen.Stadtmueller}@kit.edu

Abstract. An increasing amount of data is published and consumed on the Web according to the Linked Data paradigm. In consideration of both publishers and consumers, the temporal dimension of data is important. In this paper we investigate the characterisation and availability of temporal information in Linked Data at large scale. Based on an abstract definition of temporal information we conduct experiments to evaluate the availability of such information using the data from the 2011 Billion Triple Challenge (BTC) dataset. Focusing in particular on the representation of temporal meta-information, i.e., temporal information associated with RDF statements and graphs, we investigate the approaches proposed in the literature, performing both a quantitative and a qualitative analysis and proposing guidelines for data consumers and publishers. Our experiments show that the amount of temporal information available in the LOD cloud is still very small; several different models have been used on different datasets, with a prevalence of approaches based on the annotation of RDF documents.

Keywords: temporal information, temporal annotation, linked data.

1 Introduction

The problem of managing temporal information has been deeply studied in the field of temporal databases [18] and has been more recently addressed in the World Wide Web domain [9,1]. In fact, most data-driven and Web applications need to manage temporal information in order to capture, model, explore, retrieve, and summarize information changing over time. Moreover, the amount of rapidly changing data is likely to grow in the next future with the increasing publication of sensor data, which explicitly represents real-time data of evolving phenomenon over time [19,25,27]. As the information on the Web can change rapidly [4], also Linked Data on the Web[1] cannot be assumed to be static, with RDF statements frequently added to and removed from published datasets [29].

[1] http://lod-cloud.net/

P. Cudré-Mauroux et al. (Eds.): ISWC 2012, Part I, LNCS 7649, pp. 492–507, 2012.

As a consequence, change management and temporal information are receiving an increasing attention in the Linked Data domain. In particular, a number of significant issues have been investigated: a resource versioning mechanism for Linked Data, which allows for publishing time-series of descriptions changing over time [7]; a method to monitor the published datasets, successfully applied to several sources [17]; the maintenance of links over evolving datasets [24].

The capability of managing temporal information plays also a crucial role in several applications and research areas. In *Semantic Data Integration*, temporal information can be used to favor the most up-to-date information when fusing data [22,23]. The analysis of temporal information can also support entity resolution in some complex scenarios where the values of the attributes considered in the matching process change over time [21]. In *Temporal Query Answering and Search*, temporal information can be used to filter out the data of interest given some temporal constraint, or to rank the results of a search engine on a temporal basis. Timelines associated with data can improve the *User Experience* by presenting information in a time-dependent order [30,1].

The capability of designing effective solutions depends on the availability of temporal information and the possibility to collect and process this information across heterogeneous datasets. For example, the modification date associated with RDF documents and extracted via HTTP protocol analysis has been used to fuse data coming from different DBpedia datasets [22]; however, this information is not available in many datasets. Understanding the current status of temporal information published as Linked Data is fundamental for the development of applications able to deal with the dynamism in the data.

In this paper we investigate temporal information published in Linked Data on the Web by analysing its availability and characterisation both from a quantitative and qualitative perspective. To the best of our knowledge, despite the proposal of several approaches to model and query temporal information in RDF [11,5,30,19], support versioning for Linked Data [24], and monitor changes [29,17], a systematic and large scale analysis in this field is still missing. Based on a more precise definition of the concept of *temporal information*, we identify a specific kind of temporal information, called *temporal meta-information* in the paper. Temporal meta-information is particularly relevant to several application domains because it associates RDF statements and graphs with information about their creation, modification and validity. Since the analysis of the whole LOD cloud is unfeasible, we use the large Billion Triple Challenge[2] (BTC) dataset for our investigation. In particular, we focus on the characterization and availability of temporal meta-information, reviewing the proposed models in the literature for modelling such information and analysing their usage in the BTC.

The analysis of the BTC corpus suggests that the availability of temporal information is still scarce, with negative consequences on the design of effective solutions leveraging temporal information at large scale. Moreover, we found that none of the models proposed to manage temporal information has been widely adopted, although temporal annotations of documents seem to prevail so

[2] `http://km.aifb.kit.edu/projects/btc-2011/`

far. Based on the results of our empirical analysis, we provide some guidelines to data publishers and consumers in order to take advantage of the representation approaches proposed so far.

The paper is organized as follows: Section 2 introduces the preliminary definitions we adopt in this paper; in Section 3 we introduce the notion of temporal information and we investigate their availability in the BTC, analysing the more frequent temporal properties and the pay-level-domain they occur in. In Section 4, we review the approaches proposed in the literature for the representation of temporal meta-information and discuss their adoption in well-known datasets. In Section 5 we conduct experiments to quantitatively investigate the adoption of these models in the LOD cloud using the BTC dataset and we discuss our findings. In section 6, we draw the conclusions.

2 Preliminaries

RDF triples and RDF graphs. Given an infinite set $\mathcal{U}$ of URIs (resource identifiers), an infinite set $\mathcal{B}$ of blank nodes, and an infinite set $\mathcal{L}$ of literals, a triple $\langle s, p, o\rangle \in (\mathcal{U}\cup\mathcal{B})\times\mathcal{U}\times(\mathcal{U}\cup\mathcal{B}\cup\mathcal{L})$ is called an *RDF triple*; s, p, o are called, respectively, the subject, the predicate and the object of the triple. An *RDF graph* G is a set of RDF triples. A *named graph* is a pair $\langle G, u\rangle$, where G is a graph and $u \in \mathcal{U}$. RDF data are often stored using the N-quad format; a quad is a quadruple $\langle s, p, o, c\rangle$ where c defines the context of an RDF triple $\langle s, p, o\rangle$; the context describes the provenance of a triple, often represented by - but not limited to - an RDF graph. An RDF triple (or simply triple in the following) is also called *statement*. We call statements and graphs also *truth-valuable RDF elements*, as they can be associated with a truth value, under an interpretation function [10].

Temporal entities. We distinguish two types of temporal entities used for representing temporal information in RDF data: *time points*, represented by a single variable t^p, and *time intervals*, represented by the standard notation $[t^b; t^e]$, where t^b and t^e represent the time points respectively beginning and ending the interval and $t^b \leq t^e$ (in this paper we do not consider representations of time where intervals are not bound by time points).

Concrete Representation of Time Points on the Web. According to well-accepted best practices, time points are represented on the Web by means of *date formats*. RFC 2616 defines three different date formats that are used in the HTTP protocol[3]. The first *datetime* format, e.g., `Sun, 07 Sep 2007 08:49:37 GMT`, is defined by the standard RFC 822 [6] and is the most preferred. The second *datetime* format, e.g., `Sunday, 07-Sep-07 08:49:37 GMT`, is defined by the standard RFC 850 [15]. The third datetime format, e.g., `Sun Sep 7 08:49:37 2007`, is defined by ANSI C's *asctime* format. ISO 8601 defines a numerical date format [16]; an example of date according to this format is 2007-09-07T08:49:37.sZ. Based on this standard, dates can be also modelled as primitive datatypes in XML Schema [8]. The primitive types, `date`, `dateTime`,

[3] `http://www.ietf.org/rfc/rfc2616.txt`

`gYearMonth`, `gYear`, `gMonthDay`, `gDay` and `gMonth` defined by these specifications are usually used in RDF data. An alternative representation of time for Linked Data, which denotes temporal entities with URIs and makes use of the OWL Time ontology [12] has also been proposed [5].

RDF statements and documents. Some URIs occurring in RDF statements denote resources that are, in fact, documents (e.g., XML documents, PDF documents, or HTML pages). For the purpose of this paper it is relevant to distinguish between *generic documents* and documents publishing RDF data, called *RDF documents* in the following; like other generic documents, RDF documents can be described *by* RDF descriptions, but differently from other documents, they also contain truth-valuable RDF elements (statements and graphs). In other words, a description about an RDF document can provide a meta-description about the content of the RDF document[4].

3 Temporal Information and Temporal Properties

In this section, we first propose an abstract definition of temporal information by introducing the concept of temporal meta-information. Then we analyse the availability of temporal information in Linked Data and the properties that are used more often to represent such information.

- **Temporal information.** At the abstract level a *temporal information* can be described as a ternary relation $T(x, a, t)$, where x is a resource, a statement, or a graph, a is a property symbol, and t is a temporal entity. We call *temporal property* any property symbol used in a temporal information. Since a temporal information $T(x, a, t)$ can be also interpreted as a temporal annotation associated with the element x, the terms temporal information and temporal annotation will be used interchangeably, depending on the context.
- **Temporal meta-information.** We observe that, according to the above definition, truth valuable and non truth valuable RDF entities can be associated with temporal information. Therefore, we introduce a new concept that specifically refers to temporal information associated with truth-valuable elements: a temporal information $T(x, a, t)$ is a *temporal meta-information* if and only if x is a truth-valuable RDF element. The concept of temporal meta-information, which is defined according to semantic criteria, allows distinguishing between temporal information associated with objects in a domain of interest (e.g. the birth date of a person, but also the creation date of a PDF document) and temporal information associated with truth-valuable RDF elements (e.g, the temporal validity of statement, or the last update of an RDF document).

[4] An increasing number of RDF descriptions are also available in the RDFa syntax from plain HTML and XHTML documents; however, in this paper we focus only descriptions available in RDF/XML documents because the crawled data of the BTC corpus, which we use in our analysis, do not include data extracted from RDFa sources.

3.1 Dataset and Experimental Setup

To give more insights about the usage of temporal information in Linked Data cloud, we analyse the latest release of the BTC dataset which was crawled from the Web in May/June 2011 using a random sample of URIs from the BTC 2010 dataset as seed URIs. The BTC corpus which represents only a part of all available Linked Data on the Web, contains over 2.1 bn statements in N-Quads[5] format with over 47 K unique predicates, collected from 7.4 M RDF documents. However, our corpus constitutes a large collection of documents sampled from a wide variety of Linked Data publishers. A crawling-based approach is per design biased towards datasets that are well-interlinked, while more isolated datasets are less likely to be found. We also observe that the corpus is static, and it samples only RDF/XML, not covering data in other syntaxes like RDFa. We expect these aspects not to have any negative effects on the findings of our analysis, which still targets specifically prominent and well interlinked part of the LOD cloud.

Considering the size of the corpus, we use Apache Hadoop[6] to analyse the data. Hadoop allows for the parallel and distributed processing of large datasets across clusters of computers. We run the analysis on the KIT OpenCirrus[7] Hadoop cluster. For our analysis we used 54 work nodes, each with a 2.27 GHz 4-Core CPU and 100GB RAM, a setup which completes a scan over the entire corpus in about 15 minutes.

3.2 General Analysis

To gather a broad selection of temporal information in BTC, we employ a string-based search method which implements a class named SimpleDateFormat[8] in Java. We are confident about the correctness of the collected data because the time parser is well-known and used by a large community.

We assume that if temporal information is present, it is contained in the object position of quads. Thus, we use regular expressions to identify temporal information in the object of every quad in the BTC. However, it has been recently shown that the best practices used to publish data on the Web [3] are not always followed by publishers [13].

We notice that often RDF publishers do not use the date formats defined by standards such as RFC 822, ISO 8601 or XML Schema. In order to collect all temporal information that is represented in the BTC but is not fully compliant to standard date formats, we consider variations of the standards. The variations of the standard date formats are expressed by regular expressions based on the following patterns: `(EEE), dd MMM yy (HH:mm:(ss) (Z|z))` and `yyyy-MM-(dd('T'HH:mm:(ss).(s)(Z|z)))` respectively[9]. We extract

[5] `http://sw.deri.org/2008/07/n-quads/`
[6] `http://hadoop.apache.org/`
[7] `https://opencirrus.org/`
[8] `http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html`
[9] The value in the parentheses is optional.

Table 1. Top twenty PLDs with respect to temporal quads

PLD	quad. (M)	Tquad (K)	doc (K)	Tdoc (K)
scinets.org	56.2	3,391	51.9	44.3
legislation.gov.uk	33.1	1,249	246.4	246.4
ontologycentral.com	55.3	1,029	4.6	4.4
bibsonomy.org	34.5	881	234.7	177.3
loc.gov	7.8	854	345.3	302.9
bbc.co.uk	6.3	679	173.5	83.6
livejournal.com	169.8	530	239.2	238.9
rdfize.com	37.6	495	204.7	204.6
data.gov.uk	13.8	479	178.8	91.9
dbpedia.org	28.4	423	596.6	124.1
musicbrainz.org	2.5	359	0.3	0.3
tfri.gov.tw	153.3	272	154.4	78.2
archiplanet.org	16.3	186	79.2	53.5
freebase.com	27.8	173	572.9	109.1
vu.nl	6.8	156	294.2	26.7
fu-berlin.de	5.7	139	291.6	37.4
bio2rdf.org	20.2	129	744.7	71.6
blogspace.com	0.9	124	0.2	0.2
opera.com	24.1	124	160.3	124.1
myexperiment.org	1.5	114	26.1	13.7

Table 2. Top twenty temporal properties wrt. temporal quads

Temporal Property	quad (M)	doc (K)
dcterms:#modified	3.4	44
dcterms:modified	2.3	842
dcterms:date	1.5	247
dc:date	1.4	188
dcterms:created	0.6	450
dcterms:issued	0.2	222
lj:dateCreated	0.2	238
swivt:#creationDate	0.2	197
lj:dateLastUpdated	0.22	225
wiki:Attribute3ANRHP _certification_date	0.18	53
tl:timeline.owl#start	0.17	31
tl:timeline.owl#end	0.15	24
bio:date	0.14	143
po:schedule_date	0.14	15
swrc:ontology#value	0.096	37
cordis:endDate	0.078	0.002
nl:currentLocationDateStart	0.076	26
po:start_of_media_availability	0.074	10
foaf:dateOfBirth	0.068	68
liteco:dateTime	0.062	62

12,863,547 *temporal quads*, i.e., quads containing a temporal entity, and 1,670 unique temporal properties from the corpus.

Furthermore, to provide a deeper analysis of the distribution of temporal information within the dataset, we extract all the pay-level domains (PLDs) occurring in the context of the quads. Herein, we use PLDs to distinguish individual data providers [20]. Table 1 lists the top 20 PLDs publishing the largest number of temporal quads. For each PLD we report: the total number of quads (*quad.* in Table 1), the number of temporal quads (*Tquad.*), the number of documents (*doc*) and the number of temporal documents (*Tdoc*).

We can notice that although `scinets.org` is listed on top of the list, it does not provide the highest ratio of temporal quads over the total number of quads compared to other datasets. With respect to the temporal quads, we can notice that `musicbrainz.org` and `blogspace.com` represent the largest number of temporal quads as a proportion of all quads. Similarly for the documents, we notice that `legislation.gov.uk`, `rdfize.com` and `blogspace.com` represent the three PLDs with the largest number of temporal documents as a proportion of all documents.

Table 2 lists the top 20 temporal properties that occur more frequently in the BTC, reporting the number of quads and documents they occur in. We also provide an analysis of the distribution of the top-10 most frequent temporal properties within the most significant PLDs, which is plotted in Figure 1. It can

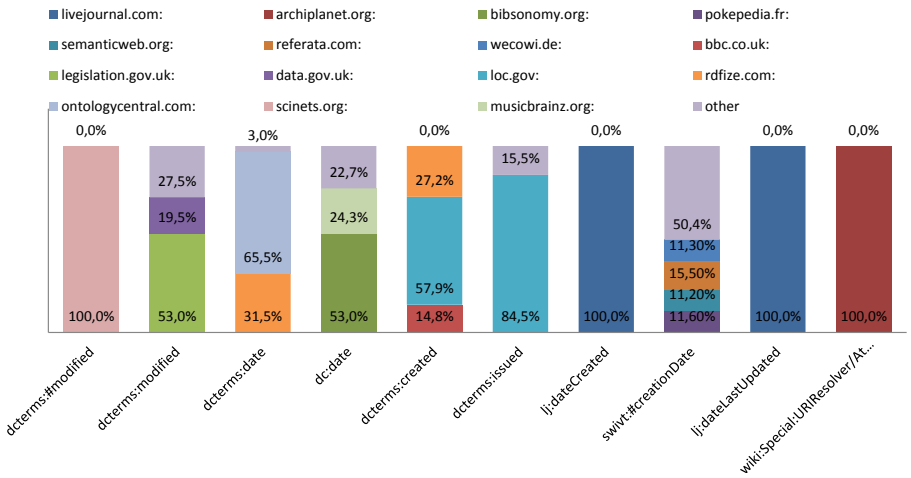


Fig. 1. Distribution of top ten temporal properties with respect to main PLDs

be noticed that not only the properties of the Dublin Core (DC) vocabulary[10] do occur much more frequently than other properties, but they are also used more often across different datasets. Remarkably, the temporal property that occurs more often in the BTC dataset, i.e., `dcterms:#modified`, has a wrong spelling (the correct spelling denotes in fact the second most frequent temporal property in the corpus). As shown in Figure 1, this is also the only temporal property published in the `scinets.org` context, and the spelling is wrong in all the quads having the same context.

4 Temporal Meta-information Description Models

In this section we focus on temporal meta-information, that is temporal information defined as T(x,a,t) where x can be either a statement or a graph. Because of the tight constraints given by the triple-based structure of RDF descriptions, the concrete RDF-based representation of an even simple temporal annotation like $T(x, a, t)$, with x being a document and t a temporal entity, requires some sophisticated mechanisms. Several approaches for providing a concrete representation of a temporal annotation have been proposed. We identify three core perspectives that have been adopted for the concrete representation of temporal meta-information in RDF:

- Document-centric Perspective, where time points are associated with RDF documents.
- Fact-centric Perspective, where time points or intervals (usually intervals) are associated with facts; since facts can be represented by one or more statements - we further separate the Fact-centric Perspective into:

[10] `http://www.dublincore.org/documents/dces/`

- Sentence-centric Perspective, which explicitly define the temporal validity of one or more statements annotating them with time points or intervals.
- Relationship-centric Perspective, which encapsulates time points or intervals into objects representing n-ary relations.

In the following we explain in detail the approaches proposed according to the aforementioned perspectives.

4.1 Document-Centric Perspective

Graphs, i.e. RDF documents, can be associated with temporal meta-information following two approaches: the first one uses HTTP-metadata, and in particular the Last-modified field of the HTTP response header; the second one expresses temporal meta-information using RDF statements with temporal properties taken from available vocabularies such as Dublin Core. Temporal meta-information following these approaches, and in particular, *Last-modified* and *ETage* properties of HTTP headers have been used for the detection of changes in Web documents publishing RDF data [29].

Protocol-based representation. A Protocol-based representation adopts point-based time modelling; the temporal meta-information is not persistently associated with a Web document, but can be extracted from the HTTP header returned in response to an HTTP GET request for the document. The temporal meta-information associates a time point, represented by a date, with a Web document G using a predicate a defined in the HTTP protocol according to the schema defined as follows:

```
HTTP Response Header
Status: HTTP/1.1 200 OK
```
a `:` t^p

Metadata-based representation. Let $\langle s, p, o\rangle$ be a statement, u_G a named graph, a_G a temporal property, t^p a time point; the Metadata-based representation associates a temporal meta-information with an RDF document as follows:

$\langle s, p, o, u_G\rangle$
$\langle u_G, a_G, t^p, u_G\rangle$

Examples of datasets providing temporal meta-information to the documents are: Protein knowledge base (UNIPROT) and legislation.gov.uk.

4.2 Fact-Centric Perspective

In the Fact-centric Perspective facts are associated with temporal meta-information that constrain their temporal validity. The first RDF model proposed to formally capture this idea is Temporal RDF [11]. In this model, RDF statements are annotated with time intervals constraining their temporal validity; the

intervals are interpreted over a point-based, discrete and linearly ordered temporal domain.

Temporal RDF-based representation. Let $\langle s, p, o\rangle$ be an RDF statement and $[t^b; t^e]$ a time interval with a starting point t^b and an ending point t^e, a Temporal RDF-based representation is a temporal annotated statement having the form $\langle s, p, o\rangle[t^b{:}t^e]$.

The encoding of the above definition into the triple-based RDF data model is not straightforward because RDF can "natively" represents only binary relations. In order to solve this problem, several approaches for encoding the temporal validity of facts into the standard RDF syntax have been proposed. These approaches follow two perspectives that present significant differences: the Sentence-centric Perspective and the Relationship-centric Perspective.

Sentence-Centric Perspective

Two strategies are adopted to represent the temporal validity of fact adopting the Sentence-centric Perspective.

Reification-based representation. Let $\langle s, p, o\rangle$ be a statement, s^{st} an identifier of a statement, a^b_S and a^e_S two temporal properties, and $[t^b{:}t^e]$ a time interval; a Reification-based representation is defined as follows:

$\langle s^{st}, \texttt{rdf:type}, \texttt{rdf:Statement}\rangle$
$\langle s^{st}, \texttt{rdf:subject}, s\rangle$
$\langle s^{st}, \texttt{rdf:predicate}, p\rangle$
$\langle s^{st}, \texttt{rdf:object}, o\rangle$
$\langle s^{st}, a^b_S, t^b\rangle$
$\langle s^{st}, a^e_S, t^e\rangle$

The first four sentences encode the reification of the statement representing the fact using the RDF vocabulary. The temporal properties a^b_S and a^e_S link the statements respectively to the beginning and the ending point of the time interval $[t^b{:}t^e]$ associated with the statement. Notice that a property a_S can have a time point or a time interval as property value. As an example of datasets adopting such approach we mention Timely Yago [30].

In the above approach, every sentence associated with a temporal annotation has to be reified. An alternative approach allows grouping together statements that have the same temporal validity by introducing the concept of *temporal graph* [28]. Temporal graphs are named graphs annotated with timeintervals; each time interval is represented by exactly one temporal graph, where all triples belonging to this graph share the same validity period. Temporal meta-information are collected in a *default graph* which occur as context in the quads.

Applied Temporal RDF-based representation. Let u_{TG} and u_G be the names respectively of a temporal graph and of the default graph, a^b_S and a^e_S two temporal properties, $[t^b{:}t^e]$ a time interval and $\langle s, p, o\rangle$ a statement; the Applied temporal RDF-based representation is defined as follows:

$\langle u_{TG}, a^b_S, t^b, u_G \rangle$
$\langle u_{TG}, a^e_S, t^e, u_G \rangle$
$\langle s, p, o, u_{TG} \rangle$

The temporal properties a^b_S and a^e_S link the temporal graph respectively to the beginning and the ending point of the time interval $[t^b{:}t^e]$. More statements can be associated with the same temporal graph. As an example of dataset that uses such approach is EvOnt [28].

Relationship-Centric Perspective

N-ary Relationship design patterns[11] are introduced to represent RDF relations with arity greater than two. These patterns model an n-ary relation with a set of RDF statements by (i) introducing a specific resource to identify the relation, and (ii) creating links between this resource and the constituents of the relation (resources and literals). These patterns can be used to associate temporal annotations with facts represented by RDF statements to constrain their temporal validity. For example, the fact "Alessandro Del Piero (ADP) plays for Juventus", which is valid within the time interval [1993,2012], can be modelled as a quintuple ⟨ADP, playsFor,Juventus,1993,2012⟩ and represented following the N-ary Relationship pattern. A resource r is introduced to identify the relation and the temporally annotated fact can be represented by the set of RDF statements ⟨ADP,playsFor,r⟩, ⟨r,team,Juventus⟩, ⟨r,from,1993⟩, ⟨r,to,2012⟩. The direction of the links and the strategies adopted for naming the properties can change according to different variants of the pattern [19,25]. However, the temporal annotations are linked to the resources that identify a relation in all the proposed variants. In this paper we define the N-ary Relationship-based representation adopting the variant described in the second use case of the W3C document, the one that occurs more frequently in the BTC corpus.

N-ary-relationship-based representation. Let $\langle s, p, o \rangle$ be an RDF statement, r a new resource, p_1 and p_2 two properties, a^b_R and a^e_R two temporal properties, and $[t^b{:}t^e]$ a time interval; the N-ary-relationship-based representation is defined as follows:

$\langle s, p_1, r \rangle$
$\langle r, p_2, o \rangle$
$\langle r, a^b_R, t^b \rangle$
$\langle r, a^e_R, t^e \rangle$

Although p_1 and p_2 can be two new properties, one of the two is usually equal to p as in the example discussed above. As an example of dataset we mention Freebase[12].

A second approach to model temporal meta-information according to the Fact-centric perspective is based on the concepts of *fluent* and *timeslice* [31]. Fluents

[11] http://www.w3.org/TR/swbp-n-aryRelations/
[12] http://www.freebase.com/

are properties that hold at a specific moment in time, i.e., object properties that change over time. The properties representing fluents link two timeslices, i.e., entities that are extended through temporal dimensions.

4D-fluents-based representation. Let $\langle s, p, o\rangle$ be an RDF statement, a_R^b and a_R^e two temporal properties, $[t^b{:}t^e]$ a time interval, and s^t and o^t two timeslices associated respectively with s and o; the 4D-fluents-based representation is defined as follows:

$\langle s^t$, `rdf:type`, `:TimeSlice`$\rangle$
$\langle s$, `:hasTimeslice`, $s^t\rangle$
$\langle s^t, a_R^b, t^b\rangle$
$\langle s^t, a_R^e, t^e\rangle$
$\langle o^t$, `rdf:type`, `:TimeSlice`$\rangle$
$\langle o$, `:hasTimeslice`, $o^t\rangle$
$\langle o^t, a_R^b, t^b\rangle$
$\langle o^t, a_R^e, t^e\rangle$
$\langle s^t, p, o^t\rangle$

Although we could not find any dataset adopting this approach, well-known ontologies like PROTON[13] and DOLCE[14] adopt it.

5 Quantitative and Qualitative Analysis

In this section we analyse and evaluate the adoption of the approaches for representing temporal meta-information. Our quantitative analysis is augmented by a qualitative discussion in Section 5.3, based on both experiments and literature, to highlight the advantages and disadvantages of each approach.

Please observe that some approaches cannot be detected automatically in the data. Therefore, for certain constructs we select a random sample and manually identify the constructs in the sample. We then scale the resulting measure to the entire dataset, which consists of 2.1bn quads in 7.4M documents. Of those, 12.8M were temporal quads (containing a date literal) occurring in 2.5M documents.

Analysing larger samples is infeasible due to the high manual effort involved in checking for constructs in the entire dataset; please note that random sampling is an established method for estimating properties of large populations (e.g., the prediction of election outcomes use small samples and achieve sufficient accuracy [2]). For instance, the error bound for Protocol-based representation is +/- 1.9%. The samples used in the experiments are available online[15].

Not all surveyed approaches are adopted on the web. We did not find any uses of the Applied temporal RDF-based representation and the 4D-fluents-based representation in the data. Table 3 gives an overview of our findings.

[13] `http://proton.semanticweb.org/`
[14] `http://www.loa.istc.cnr.it/DOLCE.html`
[15] `http://people.aifb.kit.edu/sts/data/`

Table 3. Temporal meta-information representation approaches and the respective occurrence compared to i) quads having temporal information; ii) overall quads in the BTC; iii) overall documents in the BTC (n/a = not applicable, - = no occurrence).

Perspective	Approach	Occurrence temp. quads	Occurrence overall quads	Occurrence overall docs
Document	Protocol	n/a	n/a	9.5%
	Metadata	5.1%	0.00019%	0.56%
Fact	Reification	0.02%	0.0000008%	0.006%
	Applied temporal RDF	-	-	-
	N-ary relationship	12.24%	0.0005%	0.6%
	4D-fluents	-	-	-

5.1 Document-Centric Perspective

To identify the use of the *Protocol-based representation* we ascertain how many of the URIs that identified documents in the BTC return date information in the HTTP header. We generate a random sample of 1000 documents (from the context of the quads), and for each document URI in the sample we perform an HTTP lookup to check the last-modified header in the HTTP response. We found that only 95 out of 1000 URIs returned last-modified headers.

To identify the use of the *Metadata-based representation*, we select a sample of 1000 URIs that appear in the subject position of quads with temporal information. We need to ensure that those subject URIs are in fact documents (information resources), as the Metadata-based representation pattern is concerned with documents. Thus, from the sample we exclude URIs containing the # symbol (as URIs with a # per definition do not refer to a document).

For the remaining URIs we send an HTTP request and analyse the response code to determine whether the URI identified a document. We found that 432 (43.2%) identified documents (i.e., directly returned a 200 OK status code). These information resources are not limited to RDF but they also include resources in other formats such as HTML, MP3, XML or PDF. We manually check for RDF documents with only the temporal meta-information such as modified and updated, which resulted in 51 documents.

Of the 51 RDF documents with temporal meta-information in HTTP headers, 43 are also associated with metadata-based dates. Thus, for each of the 43 identified documents we compared protocol-based last-modified and metadata-based last-modified dates. We found that protocol-based last-modified dates are more up-to-date compared to metadata-based dates with an average of almost a year (364 days).

5.2 Fact-Centric Perspective

We analyse the *Reification-based representation* in the BTC by looking for how often reified statements contain temporal information. The pattern first identifies the quads containing predicates that are defined in the RDF reification

vocabulary (i.e., `rdf:subject`, `rdf:predicate`, and `rdf:object`). From the identified cases we extract only those reified statement that have temporal meta-information associated with their subjects. In the entire BTC dataset we found 2,637 reified statements containing temporal meta-information.

To account for *N-ary-relationship-based representation* we again use a combination of sampling of the results of a query over the dataset with manual verification since n-ary relations are impossible to identify just by analysing the graph structure. Hence, we sample and manually identify occurrences.

The following pattern identifies for each document triples of the form $\langle s, p, o \rangle$ and $\langle o, p^*, o^* \rangle$ and furthermore identifies whether o is also associated with a temporal entity. Notice that the possibility to join two triples x and y where $x.object = y.subject$ is a necessary, but not sufficient condition, to identify n-ary relations. All results are contained in a set that we name *scoped set* consisting of 7M temporal quads. Hence, from the scoped set, we select three different random samples of 100 triples and we manually verify if respective documents identify an n-ary relation. Results of such manual analysis show that 10, 10 and 12 out of 100 triples in the samples are used with an n-ary relation.

5.3 Discussion and Recommendations

In the following we discuss the results and provide recommendations for data publishers and consumers.

The approaches that are part of the Document-centric Perspective are more extensively adopted than the approaches of the Fact-centric Perspective. As we hypothesised, the number of temporal meta-information associated with documents is greater than those associated with facts. Still, the use of temporal meta-information for documents (about 10% overall) are not sufficiently high enough to support our outlined use case.

We identify two approaches used for annotating documents with temporal meta-information: the ***Protocol-based representation*** and the ***Metadata-based representation***. We notice that the number of temporal meta-information are much more available in the Protocol-based rather than the Metadata-based representation. The temporal meta-information in the HTTP header, when available, are more up-to-date than the ones in the RDF document itself. *Consumers:* The applications that consume temporal meta-information should first check for temporal meta-information in the Protocol-based representation because they are more up-to-date; in case this information is not available the applications should be able to check in the Metadata-based representation. *Publishers:* Publishers should carefully update the temporal meta-information whenever the data in the document is changed; temporal meta-information in both Protocol- and Metadata-based representation should be consistent.

We identify four approaches used for annotating facts with temporal meta-information, grouped into the Sentence-centric Perspective and the Relationship-centric Perspective. These approaches associate validity expressed as temporal entities to facts.

The use of the ***Reification-based representation*** show a high complexity w.r.t. query processing [14]. The approach appears only in a very small number of quads. *Consumers:* Consumers should be able to evaluate based on the application scenario (e.g., the expected types of queries) if it is possible to either build their applications over such representation or to choose a different, and more efficient approach (e.g. Applied temporal RDF-based representation). *Publishers:* Publishers should be aware that best practices discourage the use of Reification-based representations, as they are cumbersome to use in SPARQL queries [3], even though they may be useful for representing temporal meta-information.

The performance of ***Applied temporal RDF-based representation*** has been reported to have still some efficiency issues [28], especially in the worst case, when the number of graphs (which are associated with temporal annotations) is almost equivalent to the number of triples. *Consumers:* Although we found no usage of the Applied temporal RDF-based representation in the BTC, the approach should deserve more attention because it supports expressive temporal queries based on τ-SPARQL, and can be applied to datasets that provide temporal information according to a Reification-based representation. *Publishers:* Publishers should take into consideration the worst case when using the Applied temporal RDF-based representation. Therefore, they should use it only when it is possible to group a considerable number of triples into a single graph.

The ***N-ary-relationship-based representation*** embeds time in an object that represents a relation. In the BTC, 0.6% of documents contain at least one case of N-ary-relationship-based representation, which is greater than the Reification-based representation but still represents only a small fraction of the overall number of documents. *Consumers:* Consumer applications can evaluate the temporal validity of facts from representations based on this approach. The lack of a clear distinction between plain temporal information and temporal meta-information provides high flexibility, but at the same makes difficult to predict the kind of temporal information that can be leveraged and interpret its meaning. Collecting these temporal meta-information with automatic methods is not straightforward, as shown by the manual efforts required in our analysis to identify this information. *Publishers:* Many situations require temporal meta-information associated with relations that can be modelled only as complex objects. Therefore, we recommend to publishers to use N-ary-relationship-based representation for complex modelling tasks because it allows flexibility on representing temporal meta-information associated with relation.

The ***4D-fluents-based representation*** supports advanced reasoning functionalities, but, probably also because of its complexity, has not been adopted on the Web.

6 Conclusion

The key contribution of this paper is the investigation of temporal information in Linked Data on the Web, which is important for several research and application domains. As time introduces a further dimension to the data it cannot be easily represented in RDF, a language based on binary relations; as a result, several

approaches for representing temporal information have been proposed. Based on the qualitative and quantitative analysis using the Billion Triple Challenge 2011 dataset, we came to the conclusion that the availability of temporal information describing the history and the temporal validity of statements and graphs is still very limited. If the representation of temporal validity of RDF data is somewhat more complex and can be expected to be considered in specific contexts, information about the creation and modification of data can be published with quite simple mechanisms. Yet, this information would have great value, e.g., when data coming from different sources need to be integrated and fused.

As future work, we plan to develop automatic techniques for the assessment of temporal data qualities in Linked Data, such as data currency and timeliness. With the deeper understanding of temporal information gained through our present analysis, we aim to capture and process a large amount of temporal information, overcoming several limitations of preliminary work [26].

Acknowledgements. We thank Basil Ell, Julia Hoxha and Sebastian Rudolph for their valuable comments and acknowledge the support of the EC's Seventh Framework Programme FP7/2007-2013 (PlanetData, Grant 257641).

References

1. Alonso, O., Strötgen, J., Baeza-Yates, R., Gertz, M.: Temporal Information Retrieval: Challenges and Opportunities. In: 1st Temporal Web Analytics Workshop at WWW, pp. 1–8 (2011)
2. Bartlett, J., Kotrlik, I., Higgins, C.: Organizational Research: Determining Appropriate Sample Size in Survey Research. Information Technology, Learning, and Performance Journal, 43 (2001)
3. Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web. linkeddata.org Tutorial (2008)
4. Cho, J., Garcia-Molina, H.: The Evolution of the Web and Implications for an Incremental Crawler. In: The 26th VLDB, pp. 200–209 (2000)
5. Correndo, G., Salvadores, M., Millard, I., Shadbolt, N.: Linked Timelines: Temporal Representation and Management in Linked Data. In: 1st International Workshop on Consuming Linked Data at ISWC (2010)
6. Crocker, D.H.: Standard for the Format of ARPA Internet Text Messages. RFC 822 (1982)
7. De Sompel, H.V., Sanderson, R., Nelson, M.L., Balakireva, L., Shankar, H., Ainsworth, S.: An HTTP-Based Versioning Mechanism for Linked Data. In: 3rd Linked Data on the Web Workshop at WWW (2010)
8. Fallside, D.C., Walmsley, P.: XML Schema Part 0: Primer Edition, 2nd edn. World Wide Web Consortium (2004)
9. Grandi, F.: Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web. SIGMOD Record, 84–86 (2004)
10. Gutierrez, C., Hurtado, C., Mendelzon, A.O., Pérez, J.: Foundations of Semantic Web Databases, pp. 520–541 (2011)
11. Gutierrez, C., Hurtado, C.A., Vaisman, A.A.: Temporal RDF. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 93–107. Springer, Heidelberg (2005)

12. Hobbs, J., Pan, F.: An Ontology of Time for the Semantic Web. Processings of the ACM Transactions on Asian Language Information, 66–85 (2004)
13. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the Pedantic Web. In: 3rd Linked Data on the Web Workshop at WWW (2010)
14. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An Empirical Survey of Linked Data Conformance. Web Semantics (2012)
15. Horton, M.R.: Standard for Interchange of USENET Messages. RFC 850, Internet Engineering Task Force (1983)
16. ISO 8601. Data Elements and Interchange Formats-Information Interchange-Representation of Dates and Times (2004)
17. Käfer, T., Umbrich, J., Hogan, A., Polleres, A.: Towards a Dynamic Linked Data Observatory. In: 5th Linked Data on the Web Workshop at WWW (2012)
18. Kline, N.: An Update of the Temporal Database Bibliography. SIGMOD Record, 66–80 (1993)
19. Koubarakis, M., Kyzirakos, K.: Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part I. LNCS, vol. 6088, pp. 425–439. Springer, Heidelberg (2010)
20. Lee, H.T., Leonard, D., Wang, X., Loguinov, D.: IRLbot: Scaling to 6 Billion Pages and Beyond. In: The 17th WWW, pp. 427–436 (2008)
21. Li, P., Dong, X.L., Maurino, A., Srivastava, D.: Linking temporal records. The VLDB Endowment (2011)
22. Mendes, P.N., Mühleisen, H., Bizer, C.: Sieve: Linked Data Quality Assessment and Fusion. In: 2nd International Workshop on Linked Web Data Management at EDBT (2012)
23. Panziera, L., Comerio, M., Palmonari, M., De Paoli, F., Batini, C.: Quality-Driven Extraction, Fusion and Matchmaking of Semantic Web API Descriptions. J. Web Eng. 11(3), 247–268 (2012)
24. Popitsch, N., Haslhofer, B.: DSNotify - A Solution for Event Detection and Link Maintenance in Dynamic Datasets. Web Semantics, 266–283 (2011)
25. Rodrıguez, A., McGrath, R., Liu, Y., Myers, J.: Semantic Management of Streaming Data. In: 2nd International Workshop on Semantic Sensor Networks at ISWC (2009)
26. Rula, A., Palmonari, M., Maurino, A.: Capturing the Age of Linked Open Data: Towards a Dataset-independent Framework. In: 1st International Workshop on Data Quality Management and Semantic Technologies at IEEE ICSC (2012)
27. Sheth, A., Henson, C., Sahoo, S.: Semantic Sensor Web. IEEE Internet Computing 12(4), 78–83 (2008)
28. Tappolet, J., Bernstein, A.: Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 308–322. Springer, Heidelberg (2009)
29. Umbrich, J., Hausenblas, M., Hogan, A., Polleres, A., Decker, S.: Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In: 3rd Linked Data on the Web Workshop at WWW (2010)
30. Wang, Y., Zhu, M., Qu, L., Spaniol, M., Weikum, G.: Timely YAGO: Harvesting, Querying, and Visualizing Temporal Knowledge from Wikipedia. In: The 13th EDBT, pp. 697–700 (2010)
31. Welty, C., Fikes, R., Makarios, S.: A Reusable Ontology for Fluents in OWL. In: Frontiers in Artificial Intelligence and Applications, p. 226 (2006)

[P5] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 1225–1236. 2013.

Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data

Steffen Stadtmüller
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
steffen.stadtmueller@kit.edu

Sebastian Speiser
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
speiser@kit.edu

Andreas Harth
Institute AIFB
Karlsruhe Institute of
Technology (KIT), Germany
harth@kit.edu

Rudi Studer
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
studer@kit.edu

ABSTRACT

An increasing amount of applications build their functionality on the utilisation and manipulation of web resources. Consequently REST gains popularity with a resource-centric interaction architecture that draws its flexibility from links between resources. Linked Data offers a uniform data model for REST with self-descriptive resources that can be leveraged to avoid a manual ad-hoc development of web-based applications. For declaratively specifying interactions between web resources we introduce *Data-Fu*, a lightweight declarative rule language with state transition systems as formal grounding. Data-Fu enables the development of data-driven applications that facilitate the RESTful manipulation of read/write Linked Data resources. Furthermore, we describe an interpreter for Data-Fu as a general purpose engine that allows to perform described interactions with web resources by orders of magnitude faster than a comparable Linked Data processor.

Categories and Subject Descriptors

H.5.4 [**Hypertext/Hypermedia**]: Architectures

General Terms

Languages, Performance

Keywords

REST; Linked Data; Web Interaction; Rule Language; Interpreter

1. INTRODUCTION

There is a growing offer of functionality via web APIs[1]. Increased value comes from combining data from multiple sources and functionality from multiple providers. The importance of such compositions is reflected in the constant growth of mashups – small programs that combine multiple web APIs [33]. There is a strong movement in the web community toward a resource-oriented model of services based on Representational State Transfer (REST [11]). Flexibility, adaptivity and robustness are the major objectives of REST and are particularly useful for software architectures in distributed data-driven environments such as the web [22]. However, data sources and APIs are published according to different interaction models and with interfaces using non-aligned vocabularies, which makes writing programs that integrate offers from multiple providers a tedious task.

The goal of our work is to provide a declarative means to specify interactions between data and functionality from multiple providers. Such declarative specifications provide a modular way of composing the functionality of multiple APIs. Also, declarative methods allow for automatically optimising a program and parallelising the execution.

In a REST architecture, client and server are supposed to form a contract with content negotiation, not only on the data format but implicitly also on the semantics of the communicated data, i.e., an agreement on how the data have to be interpreted [32]. Since the agreement on the semantics is only implicit, programmers developing client applications have to manually gain a deep understanding of the provided data, often based on natural text descriptions. The combination of RESTful resources originating from different providers suffers particularly from the necessary manual effort to use and combine them. The reliance on natural language descriptions of APIs has led to mashup designs in which programmers are forced to write glue code with little or no automation and to manually consolidate and integrate the exchanged data.

Linked Data unifies a standardised interaction model with the possibility to align vocabularies using RDF, RDFS and OWL. However, the interactions are currently constrained to simple data retrieval. Following the motivation to look beyond the exposure of fixed datasets, the extension of Linked Data with REST technologies has been explored [5, 34] and

[1]Alone `http://programmableweb.com/` lists 7,991 APIs on November 24th 2012, which is almost twice the number from one year earlier.

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

led recently to the establishment of the *Linked Data Platform*[2] W3C working group.

Several existing approaches recognise the value of combining RESTful services and Linked Data [17, 26, 30]. In this paper, we go one step further and propose *Data-Fu*, a data- and resource-driven programming approach leveraging the combination of REST with Linked Data. Data-Fu enables the development of applications built on semantic web resources with a declarative rule language. The main goal of Data-Fu is to minimise the manual effort to develop web-based applications and the preservation of loose coupling by

- leveraging links between resources provided by Linked Data, and
- specifying desired interactions dependent on resource states, which is enabled by a uniform state description format, i.e., RDF.

A further requirement for our programming approach in a web-based environment is a fast and scalable execution of the applications. While there has been recent work on extending the Map/Reduce model for data-driven processing [15, 4], these approaches are geared towards deployment in data centers. In contrast, our approach operates on the networked open web.

This paper is based on a previous publication on a data-driven programming model for the web [27] and describes

- how self-descriptive resources can be designed to enable loosely coupled clients (Section 4.1);
- a service model for REST based on state transition systems as formal grounding (Section 4.2);
- the Data-Fu language, a declarative rule-based execution language to allow an intuitive specification of the interaction with resources from different providers (Section 5);
- an execution engine as an artefact to perform the defined interactions in a scalable manner (Section 6).

We provide a motivating scenario in Section 2. We evaluate our approach in two ways: (i) we describe throughout the paper how our motivating scenario can be realised with Data-Fu; and (ii) we conduct performance experiments with the Data-Fu interpreter in Section 7. Section 8 covers existing work. We conclude in Section 9.

2. MOTIVATING SCENARIO

In our scenario, we consider the Acme corporation, a consumer goods producer, that aims at extending their social media activities to a broader range of dissemination channels (for more on multi-channel communication see [7]). Acme's marketing department observes that while the number of potential channels is constantly increasing, the channels can be broadly categorised into micro blog services and social networks. Information about new products, special offers, and other news should be disseminated in the following ways: (i) posts on the company's micro blogs; and (ii) messages to social network users who are followers of the company.

We assume that the dissemination channels offer Linked APIs, i.e., resources are exposed that offer read/write Linked Data functionality.[3]

[2] `http://www.w3.org/2012/ldp/charter`

[3] If there is no Linked API available, the conventional APIs can be easily wrapped to consume and produce RDF, see, e.g., [29, 17]. Wrapping APIs is out of scope of this paper.

Table 1: URI prefixes used throughout this paper

Prefix	IRI
`acme:`	`http://acme.example.org/company/`
`p:`	`http://acme.example.org/vocabulary/`
`sna:`	`http://sna.example.org/lapi/`
`snb:`	`http://snb.example.org/rest/`
`mb:`	`http://mb.example.org/interface/`

The marketing department orders a system from Acme's IT that manages the dissemination channels and automatically disseminates a post to all available channels either as a micro blog entry or as a personal message. Initially the micro blog service MB and the social network SNA have to be supported. Marketing will supply their posts in an Acme-specific vocabulary as so-called InfoItems.

After a while, the marketing department decides to add the new social network SNB as a dissemination channel, which requires two steps: (i) the IT department extends the dissemination system to support the interface of SNB; and (ii) the marketing department adds Acme's identity in SNB to the dissemination channels.

Throughout the paper, we will illustrate our technical contributions by realising bits and pieces of the proposed scenario. When modeling services and interactions, we will use a number of URI prefixes for brevity that are either common[4] or listed in Table 1.

3. BACKGROUND

According to the Richardson maturity model [24] REST is identified as the interaction between a client and a server based on three principles:

- The use of URI-identified resources.
- The use of a constrained set of operations, i.e., the HTTP methods, to access and manipulate resource states.
- The application of hypermedia controls, i.e., the data representing a resource contains links to other resources. Links allow a client to navigate from one resource to another during his interaction.

The idea behind REST is that applications, i.e., clients, using functionalities provided on the web, i.e., APIs, are not based on the call of API-specific operations or procedures but rather on the direct manipulation of exposed resource representations or the creation of new resource representations. A resource can be a real world object or a data object on the web. The representation of a resource details the current state of the resource. A manipulation of the state representation implies that the represented resource is manipulated accordingly. For brevity in this paper we often talk about "the manipulation of a resource", when we actually mean "the manipulation of the state representation of a resource and the subsequent change of the resource itself".

The flexibility of REST results from the idea that client applications do not have to know about all necessary resources. The retrievable representations of some known resources contain links to other resources, that the client can

[4] See `http://prefix.cc/` for their full URIs, accessed on November 22nd 2012.

discover during runtime. Clients can use such discovered resources to perform further interaction steps.

The Linked Data design principles[5] also address the use of URI-identified resources and their interlinkage. However Linked Data is so far only concerned with the provisioning and retrieval of data. In contrast to REST, Linked Data does distinguish explicitly between URI-identified objects (i.e., non-information resources) and their data representation (information resources). An extension of Linked Data with REST to allow for resource manipulation leads to read/write Linked Data, i.e., information resources can be accessed and manipulated. REST furthermore implies that a change of an information resource implies a change in the corresponding non-information resource.

The development of applications in a REST framework is especially challenging, since the links between resources and the resource states can only be determined during runtime, however, programmers have to specify their desired interactions at design time.

Traditional service composition approaches that aim to decrease the manual effort to use web-offered functionality lead to a tight coupling between client and server, i.e., they sacrifice flexibility and are prone to failures due to server-side changes. Traditional composition approaches often fail to leverage links between resources and do not provide straightforward mechanisms to dynamically react to state changes of resources. The reaction on state changes becomes especially important in a distributed programming environment, since a client cannot ex ante predict the influence of other clients on the resources, i.e., REST does not allow a client to make assumptions on resource states.

4. READ/WRITE LINKED DATA

In this section, we describe our approach for modelling of RESTful services based on Linked Data. Our approach has two layers:

- Individual *Read/Write Linked Data Resources* with descriptions that allow predicting the effect of the execution of a functionality before invocation (Section 4.1);
- A formal *REST Service Model*. A single REST service can consist of several resources, potentially spread over different servers. The *service model* is the grounding for describing the interactions that are offered by the individual RESTful Linked Data resources and the overall service (Section 4.2).

4.1 Read/Write Linked Data Resources

In a RESTful interaction with Linked Data resources only the HTTP methods can be applied to the resources. The semantics of the HTTP methods itself is defined by the IETF[6] and do not need to be explicitly described.

Table 2 shows an overview of the most important HTTP methods. We can distinguish between safe and non-safe methods, where safe methods guarantee not to affect the current states of resources. Further, some of the methods require additional input data to be provided for their invocation. The communicated input data can be subject to requirements that need to be described to allow an automated interaction, e.g., the input data can be required to use a specific vocabulary. Furthermore, the effect of a non-safe method on the state of an addressed resource can depend on the input data. The dependency between communicated input and the resulting state of resources also needs to be described. Therefore, only the non-safe HTTP methods that require input data need further description mechanisms. Note, the POST method can also influence the states of not directly addressed resources. The precise effect of a POST depends on the resource, since POST allows to send input data to a data-handling process of a resource.

Table 2: Overview of HTTP methods

Method	Safe	Input required	Intuition
GET	x		Retrieve the current state of a resource.
OPTIONS	x		Retrieve a description of possible interactions.
DELETE			Delete a resource
PUT		x	Create or overwrite a resource with the submitted input.
POST		x	Send input as subordinate to a resource or submit input to a data-handling process.

The state of a Linked Data resource is expressed with RDF. It is sensible to serialise the input data in RDF as well, i.e., data that is submitted to resources to manipulate their state. To convey the resulting state change after application of a HTTP method we use RDF output messages. In previous work [20] we analysed the potential of graph patterns, based on the syntax of SPARQL[7], to describe required input as well as their relation to output messages. The resulting graph pattern descriptions are attached to the resource and can be retrieved via the *OPTIONS* method on the respective resource. Therefore the resources stay self-descriptive, i.e., their current state can be retrieved with *GET*, the possibilities to influence their state with *OPTIONS*.

Example. Acme's IT creates the resource `acme:Acme` representing Acme. A GET on `acme:Acme` returns the following initial description: `acme:Acme rdf:type p:Company .` The marketing department updates the `acme:Acme` resource with the dissemination channels SNA and MB by performing a PUT with the following input data:

```
acme:Acme  rdf:type       p:Company .
acme:Acme  p:dissChannel  sna:Acme, mb:Acme .
sna:Acme   rdf:type       p:SocialNetworkID .
mb:Acme    rdf:type       p:MicroBlogTimeline .
```

A subsequent GET on `acme:Acme` would result in exactly the description that marketing supplied with their PUT request.

A GET on `sna:Acme`, Acme's identifier in the social network SNA, would result in a description of Acme in SNA's vocabulary including its fans:

```
sna:Acme  rdf:type      sna:CommercialOrganisation .
sna:Acme  sna:founded   "11/20/2012" .
sna:Acme  sna:hasFan    sna:User1, sna:User2, ... .
```

The resources representing users in the SNA network provide

[5] http://www.w3.org/DesignIssues/LinkedData.html
[6] http://www.ietf.org/rfc/rfc2616.txt
[7] http://www.w3.org/TR/rdf-sparql-query/#GraphPattern

functionality to send messages to the corresponding users. A POST can be employed to send a message to a user resource (e.g., to `sna:User1`). The input data for the POST contains its `sna:sender` and its `sna:content`, according to the description of the user resource that can be retrieved with an OPTIONS request:

```
INPUT:   ?m  rdf:type       sna:Message .
         ?m  sna:sender     ?s .
         ?m  sioc:content   ?c .
OUTPUT:  ?m  sna:sender     ?s .
         ?m  sioc:content   ?c .
         ?m  sna:receiver   sna:User1 .
```

Acme's timeline `mb:Acme` on the micro blogging service MB also supports the POST operation. Figure 1 illustrates the timeline resource `mb:Acme` of our example, with a set of entries in the current state and the graph pattern that describe how a new entry can be POSTed.

Applying a DELETE on a blog post, e.g., one that advertises an expired sale, does not require input; its effect is inherently defined by the method: the entry is erased.

4.2 REST Service Model

A REST service can be identified with the resources it exposes. An interaction within a REST architecture is based on the manipulation of the states of the exposed resources.

We develop a model, that allows to formalise the functionalities exposed by a REST API based on read/write Linked Data resources. A formal service model serves as rigorous specification of how the use of individual HTTP methods influences resource states and how these state changes are conveyed to interacting clients.

We model a Linked Data-based RESTful service as a REST state transition system (RSTS) similar to a state machine as defined by Lee and Varaiya [18]. The behavior of the clients themselves is not in the scope of this model, it rather formalises all possible interaction paths of a client with the resources.

DEFINITION 1. *A REST state transition system (RSTS) is defined as a 5-tuple* $RSTS = \{R, \Sigma, I, O, \delta\}$ *with:*

- *A set of resources* $R = \{r_1, r_2, ...\}$.
- *A set of states* $\Sigma = \{\sigma_1, ..., \sigma_m\}$. *Each state* $\sigma_k \in \Sigma$ *of the RSTS is defined as the union of the states of all resources:* $\sigma_k = \bigcup_{r_i \in R} \overline{r_i^k}$. *The state of a single resource* $r_i \in R$ *in a state* σ_k *is given by its RDF representation* $\overline{r_i^k} \in G$, *where* G *is the set of all possible RDF graphs.*
- *An input alphabet* $I = \{(r, \mu, g) : R \times M \times G\}$, *where* $M = \{GET, DELETE, PUT, POST\}$ *is the set of the supported HTTP methods*[8].
- *An output alphabet* $O = \{(c, o) : C \times G\}$, *where* C *is the set of all HTTP status codes.*
- *An update function* $\delta : \Sigma \times I \to \Sigma \times O$ *that returns for a given state and input the resulting state and the output. We decompose* δ *into a state change function* $\delta^s : \Sigma \times I \to \Sigma$ *and an output function* $\delta^o : \Sigma \times I \to O$, *such that* $\delta(\sigma, i) = (\delta^s(\sigma, i), \delta^o(\sigma, i))$. *We define the state change function as*

$$\delta^s(\sigma_k, (r_i, \mu, g)) = \begin{cases} \sigma_k, & \text{if } \mu = GET \\ \sigma_k \setminus \{\overline{r_i^k}\}, & \text{if } \mu = DELETE \\ (\sigma_k \setminus \{\overline{r_i^k}\}) \cup g, & \text{if } \mu = PUT \\ \mathsf{post}_i(\sigma_k, g), & \text{if } \mu = POST, \end{cases}$$

where the function post_i *encapsulates the resource specific behaviour of a POST request, as described by its INPUT/OUTPUT patterns, which can be obtained via an OPTIONS request on the resource. Let* σ_l *be the new state as defined by* δ^s, *we define the output function as*

$$\delta^o(\sigma_k, (r_i, \mu, g)) = \begin{cases} (c\,, \overline{r_i^k}), & \text{if } \mu = GET \\ (c\,, \emptyset), & \text{if } \mu = DELETE \\ (c\,, \sigma_l \backslash \sigma_k), & \text{if } \mu = PUT \\ (c\,, \sigma_l \backslash \sigma_k), & \text{if } \mu = POST. \end{cases}$$

A client interacting with a service modelled by an $RSTS = \{R, \Sigma, I, O, \delta\}$ creates an input $i = (r_i, \mu, g)$ for $RSTS$ by invoking the HTTP method μ on the resource r_i and passing the potentially empty RDF graph g in the request body. Depending on the current state σ_k of the service the following happens:

1. The service transitions into the state $\delta^s(\sigma_k, (r_i, \mu, g))$.
2. The client gets an HTTP response with the HTTP code c and the RDF graph g' in the body, where $(c, g') = \delta^o(\sigma_k, (r_i, \mu, g))$.

Safe methods that do not change any resource states, describe self-transitions, i.e., transitions that start and end in the same state.

The output function in the case of PUT and POST report to the client the effect the invocation of the method had on the state of the RSTS (i.e. $\sigma_l \backslash \sigma_k$).

Resources do not necessarily allow the use of all HTTP methods. Note that all state change functions are defined for every resource, i.e., every resource can be addressed with all methods: If a resource does not allow for the application of a specific method, the state change function describes a self-transition.

The defined service model serves as formal grounding of the execution language described in Section 5. However, the self-descriptive resources provide sufficient information for the interaction with the exposed resources.

- The current state of Linked Data resources – and therefore the state of the RSTS – can be accessed as RDF.
- The possible transitions and the state they result in are independent of the specific resource, except for POST transitions. The effect of POST transitions is declared with graph pattern descriptions (see Section 4.1).

Example. Figure 2 illustrates a state transition in RSTS where an entry is POSTed to `mb:Acme`. Note, that a client could derive the input for the POST method from the states of other resources (e.g., from Acme InfoItems).

5. THE DATA-FU LANGUAGE

In this section, we present Data-Fu[9], an *execution language* to instantiate a concrete interaction between a client

[8]For brevity we focus here on the four most important methods. Other methods can be added analougously.

[9]We use the name *Data-Fu* in adaption of the term *google-fu*, which adopts the suffi x*Fu* of from Kung Fu, implying great skill or mastery. Thus Data-Fu hints at the mastery of data interaction that can be achieved with the language.

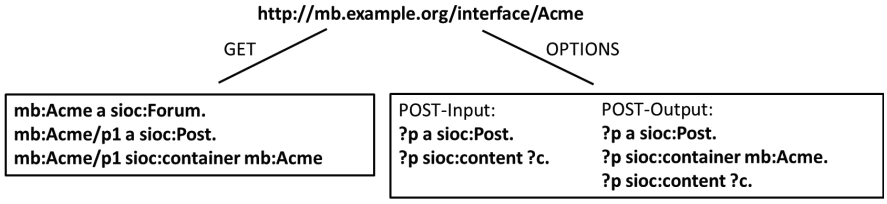


Figure 1: Self-descriptive resource: current state can be accessed with GET, input/output description with OPTIONS

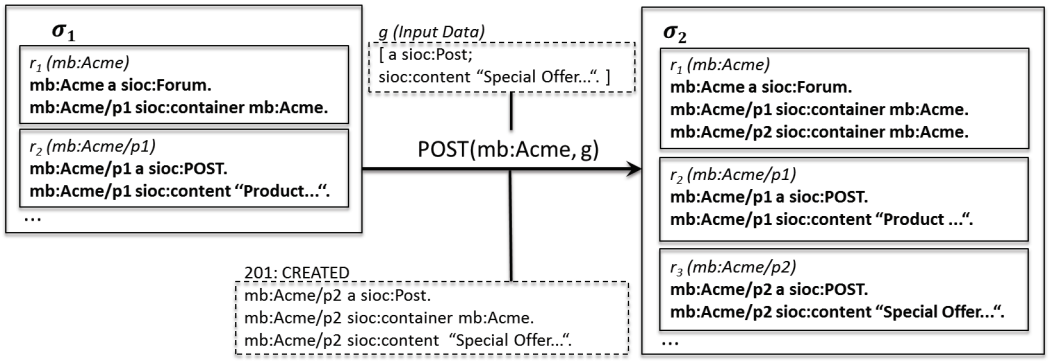


Figure 2: State transition of a RSTS, with excerpts of two states.

and resources, which preserves the adaptability, robustness and flexibility of REST.

In a resource-driven environment, applications retrieve and manipulate resources exposed on the Web. Since the resources can potentially be accessed by a multitude of clients, applications have to react dynamically on the state of the resources. Therefore, an important factor in the development of resource-driven applications is the dependency between the invoked transitions and resource states. The dependency between the invoked state transitions (i.e., applied HTTP methods) and the states of resources is that

1. input data for the transition is derived from RDF detailing the states of resources and/or
2. the transition is only invoked, if resources are in a specified state.

Data-Fu, a declarative rule-based execution language, enables programmers to define their desired state transitions. Data-Fu rules specify the interaction of a client with RESTful Linked Data resources and congruously a path through the RSTS. Further Data-Fu allows to specify the conditions under which a specific transition is to be invoked as subject to the states of resources.

Definition 2. *A rule ρ is of the form $\mu(r, g) \leftarrow q$, where $\mu \in M$ is an HTTP method, $r \in R \cup V$ is a resource or a variable with V the set of all variables, $g \in G \cup P$ is a (potentially empty) RDF graph or graph pattern, and $q \in P$ is a conjunctive query with P the set of all possible RDF graph patterns. If r is a variable, it must be bound in q. If g is a graph pattern, all its variables must be bound in q.*

The head of a rule corresponds to an update function of the RSTS in that it describes an HTTP method that is to be applied to a resource. The rule bodies are conjunctive queries that allow programmers to express their intention under which condition a method is to be applied. Thus, programmers can define an interaction pattern with a set of rules for their client applications.

The use of conjunctive queries is motivated by the idea that clients have to maintain a knowledge space (KS) in which they store their knowledge about the states of the resources they interact with [17, 25]. KS is filled with the RDF data the client receives after applying an HTTP method, as defined by the output functions of the RSTS. The output always informs the client about the current state after the application of the method.

Concretely N3 graph patterns are employed as queries q, which are evaluated over KS. If the evaluation of q is successful, i.e., matches are found in KS, the defined HTTP method μ is applied to r with input g. The query q can also be used to dynamically (i.e., during runtime)

1. derive input data from the states of other resources, as stored in KS and
2. identify the resource to which an HTTP method has to be applied, i.e., leveraging hypermedia controls.

Regarding 1: Instead of specifying the input data g explicitly as RDF graph, a graph pattern can be used. If a match is found for q in KS, the identified bindings for q are used to replace the variables in g to establish the input data for the interaction (with HTTP method μ at resource r). g as graph pattern and q act together similar to a SPARQL *construct* query over KS, where the result of the query is used as input data for the invocation of the method μ.

Regarding 2: To preserve the flexibility provided by REST our execution language has to be able to make use of links in the resource states to other resources. Rather than specifying the addressed resource r of a rule explicitly as URI, a variable can be used. If a match is found for q in KS, an identified binding for a variable q is used for the variable r. r as variable and q act together as a SPARQL *select* query to identify the targeted resources of method μ.

A Data-Fu program terminates when there are no active transitions and no rules can be activated that could trigger new transitions. In general, termination of a program cannot be guaranteed, as every transition can result in data that triggers new transitions. However, the termination of a program is not necessarily intended by a programmer, in the case of applications that are supposed to continuously interact with resources. Furthermore, the deletion and change of resources can lead to applications with a non-deterministic execution behavior. For discussions about properties of rule sets in related languages that guarantee termination and determinism, we refer the reader to [2].

Example. The IT department of Acme creates the dissemination system with four Data-Fu rules. The marketing department has simply to create new InfoItems and the system automatically distributes the information over the dissemination channels of Acme. The rules are defined as follows:

1. Whenever a InfoItem is found, retrieve the resource `acme:Acme` to get an up-to-date list of the current dissemination channels.
   GET (`acme:Acme`, {}) ← { `?x rdf:type p:InfoItem` }

2. If a `p:MicroBlogTimeline` is found (from the retrieved dissemination channels), post a new entry to the timeline using the content from the InfoItem.
   ```
   POST (?mb, { []  rdf:type      sioc:Post ;
                    sioc:content  ?c .          } )
   ← {  ?x   rdf:type   p:InfoItem .
        ?x   p:content  ?c .
        ?mb  rdf:type   p:MicroBlogTimeline   } .
   ```

3. If a social network ID of Acme is found (from the retrieved dissemination channels), retrieve the representation of Acme from the social network to get a list of Acme's followers.
   GET (`?sid`, {})
   ← { `?sid rdf:type p:SocialNetworkID` } .

4. Post to every found follower of Acme on SNA a message with the content of the InfoItem.
   ```
   POST (?f, { []  rdf:type      sna:Message ;
                   sna:sender    sna:Acme ;
                   sna:content   ?c .          })
   ← {  sna:Acme  sna:hasFan   ?f .
        ?x        rdf:type     p:InfoItem .
        ?x        p:content    ?c           }.
   ```

The described rules disseminate new information items automatically to social network SNA and the micro blog MB. IT deploys the dissemination system itself as a read/write Linked Data resource under `acme:Dissemination`. Marketing uses the dissemination service by POSTing a graph to the dissemination resource that corresponds to the following input pattern:

{ `?x rdf:type p:InfoItem. ?x p:content ?c` } .

Other dissemination channels can easily be added to the system, simply by adding corresponding rules in the system. For example, we consider that IT adds support for social network SNB by adding a rule that uses SNB's vocabulary for retrieving followers and sending a message:

```
POST (?f, {  []  rdf:type     snb:PrivateMsg ;
                 snb:origin   snb:ACME ;
                 snb:text     ?c .              })
← {  snb:ACME  snb:followedBy  ?f .
     ?x        rdf:type        p:InfoItem .
     ?x        p:content       ?c             }.
```

The new dissemination channel is active when marketing PUTs Acme's identifier in SNB's network to `acme:Acme`.

6. THE DATA-FU INTERPRETER

The Data-Fu interpreter is an execution engine for service interactions specified as a set of Data-Fu rules. The engine implements the KS as well as the functionality to invoke interactions with resources as defined in the rules. In practice, we translate a Data-Fu program into a logical dataflow network, which is then optimised (e.g., re-using triple patterns and joins). The optimised logical network is then transformed into an evaluator plan that actually implements the dataflow network.

We realise the evaluator plan for the Data-Fu engine as a streaming processor that can process several queries in parallel. We implement the processor as a multi-threaded component with one thread evaluating individual triple patterns, and separate threads for each join operator and for each rule head, i.e., the component that performs the state transitions by invoking the corresponding HTTP methods on resources. The joins are implemented as symmetric hash join operators [35]. The implemented dataflow network is similar to a parallel version of the Rete algorithm [12].

To enable a wide variety of applications the engine can include an extension to support the interaction with REST resources that are not based on Linked Data. The engine can store data entities (e.g., binaries, JSON documents) received from such services separately. A triple pointing to a received non-RDF entity can be included in KS, thus the entities can be used in the logic of the execution rules. However, an interaction with such non-RDF entities requires to fall back to a more mashup-like programming approach.

Example. The dataflow network shown in Figure 3 evaluates the plan generated for the Data-Fu program for Acme's dissemination system. We can see that joins (e.g., the join on `?x`) are re-used, i.e., have multiple outgoing edges. The triple stream is initialised by the service input, which is sent by the client via a POST request. If the input data contains a description of an information item, it will trigger the rule retrieving Acme's description containing links to its dissemination channels. The social networks will fire a rule, which then retrieves the social network id's of Acme and thus retrieve the corresponding followers. Both social network followers and micro blog timelines will then trigger the corresponding POST actions that will sent the information item in the appropriate vocabulary to the dissemination channels, i.e., as micro blog posts or personal messages to the followers.

7. EVALUATION

To evaluate the scalability of the Data-Fu engine we compared execution times for different numbers of interactions and rules with Cwm[10], a data-processor for the Semantic Web. Cwm uses a local triple store that supports the full N3 language to save data and intermediate results. The local

[10] `http://www.w3.org/2001/sw/wiki/CWM`

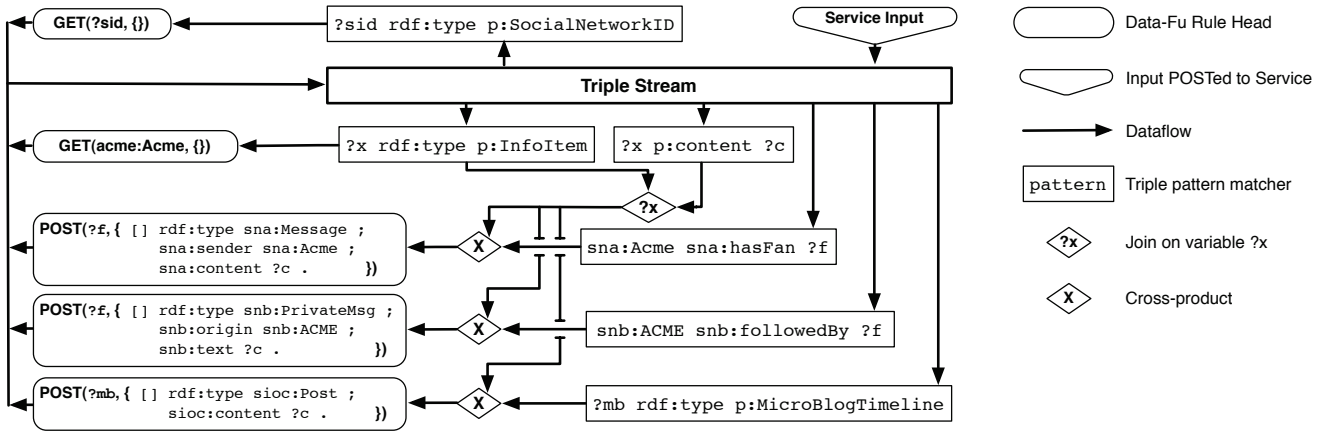


Figure 3: Dataflow network of Acme's dissemination system

triple store of Cwm uses seven indices to allow for a rapid readout of the local data with almost every combination of subject, predicate and object patterns. For inferencing Cwm uses a forward chain reasoner for N3 rules. The pattern matching for the rules is done by recursive search with optimisations, such as identifying an optimal ordering for the evaluation of the rules and patterns.

Cwm is built as a general purpose tool to query, process, filter and manipulate data from the Semantic Web. As such, the motivation behind Cwm is closest to the Data-Fu engine, compared with any other rule engines or reasoning systems, to the best of our knowledge. However, Cwm is not targeted on the direct RESTful manipulation of web resources, but their retrieval and the local manipulation of the data. Therefore to make the systems comparable we limit the evaluated interactions to GET transitions, i.e., we use only rules that retrieve resources, if a match for the rule body is found. Please note that the limitation to GET transitions does not influence the validity of the evaluation: Since additional execution time when using non-safe interactions (e.g., PUT, POST) only results from time required to transmit data to resources and the subsequent time necessary to process this data by the server, where the resource resides. This time overhead caused by non-safe transitions is neither influenced by the Data-Fu engine, nor could it be avoided by any other system that we could use as comparison.

We conducted the experiments on a 2.4 GHz Intel Core 2 Duo with 4 GB of memory (2 GB assigned to Java virtual machine on which the experiments run). Thus we evaluate the Data-Fu engine on commodity hardware with the intent to show the parallelisation-based scalability of the Data-Fu engine not only on high-end industrial machines.

We deploy Linked Data resources used for the interactions locally on an Apache Tomcat[11] server to further minimise execution time variations caused by establishing HTTP connections and retrieving data over the web. In the rules used by the Data-Fu engine and Cwm the resources are addressed with their localhost address. Every deployed resource represents a number. Every number resource is typed as `number` and contains its value as literal and a link to the successor of the number:

```
local:1  rdf:type          local:number.
local:1  local:value       "1".
local:1  local:successor   local:2.
```

We chose this design to easily keep track of the number of performed interactions.

For the evaluation we start with the resource number 0, which we manually inject into the Data-Fu engine and Cwm. We identify and retrieve the successor of the number. The successor of a number yields a new successor to retrieve, and so on. The interactions of this set-up are illustrated in Figure 4.

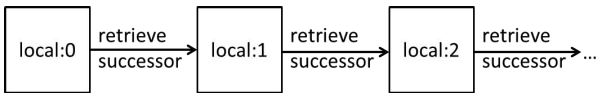


Figure 4: Interactions of evaluation set-up with one rule

We realise the interactions with the Data-Fu Engine and Cwm (for the latter in two different ways) as follows:

- *Data-Fu:* For the Data-Fu engine we use a rule:

```
GET (?suc, {})  ←  {?n rdf:type local:number
                    ?n local:value ?v
                    ?n local:successor ?suc}
```

  The rule body queries for a resource (variable `?n`) that is typed as number, has a value and a successor. If a match is found, a GET transition is triggered at whatever URI is identified to be the successor of the matched number. The Data-Fu engine adds the retrieved representation of the successor to the data flow network, which results in the identification of the next successor to retrieve. Thus, all numbers are iteratively found and retrieved.
- *Cwm direct:* Cwm offers built-in functions to perform web-aware queries in rules. The keyword `log:semantics` in a query of a rule allows to resolve a URI and bind the retrieved RDF data to a variable as formula. The formula bound to a variable can then be used to construct triples in the rule head. We used the following rule to perform the desired interaction:

[11] `http://tomcat.apache.org/`

```
{{ :n rdf:type local:number.
   :n local:value :v.
   :n local:successor :suc }
        local:is local:known. }
:suc log:semantics :sem.
⇒
  { :sem local:is local:known. }
```

Like in the approach for the Data-Fu engine we query for the successor of a number. The successor is retrieved and bound as formula in subject position to a new triple that is written to the triple store. Since the retrieved representation of the number appears only as formula in triples we have to extend the query in the rule body to search for the successor of a number in a formula in subject position of a triple, thus making the query slightly more complicated than in the case of the Data-Fu engine. Cwm repeatedly applies the rule to the triple store, thus retrieving all numbers.

- *Cwm import:* To compare the performance of Cwm with the Data-Fu engine, where the queries of the rules are equally complex, we implemented the desired retrieval with another approach, with the following rule:

```
{ :n rdf:type local:number.
   :n local:value :v.
   :n local:successor :suc }
⇒
  { :n owl:imports :suc. }
```

We use the same query to identify the successor of a number as for the Data-Fu engine. For every found match we write a triple to the Cwm store, that marks the identified successor with `owl:imports`. Cwm offers a command to retrieve all resources marked with owl:imports. This allows us to programmatically instruct Cwm to apply the rule and retrieve the successor, as many times as needed. Note, that this implementation of the interaction does not deliver the same functionality as with the Data-Fu engine: We have to manually define how often the rule followed by the retrieve command is to be applied (once for every number), rather then having the engine automatically retrieve all the numbers.

We evaluate the execution time of the interaction with all three setups for sets of 20, 40, 60, 80 and 100 numbers. With the approaches *Data-Fu* and *Cwm direct* the interaction ends when the last number in a set does not refer to a next successor to retrieve. For *Cwm import* we had to decide manually how often the rule is applied and thus how many numbers are retrieved and when the interaction stops. The results are shown in Table 3 and Figure 5. We provide the average execution times from ten runs to reduce variations.

Table 3: Average execution time from ten runs for different evaluation set-ups with one rule

number set size	Data-Fu	Cwm direct	Cwm import
20	342 ms	1549 ms	468 ms
40	371 ms	5144 ms	976 ms
60	500 ms	11272 ms	1595 ms
80	555 ms	21005 ms	2309 ms
100	594 ms	32213 ms	3688 ms

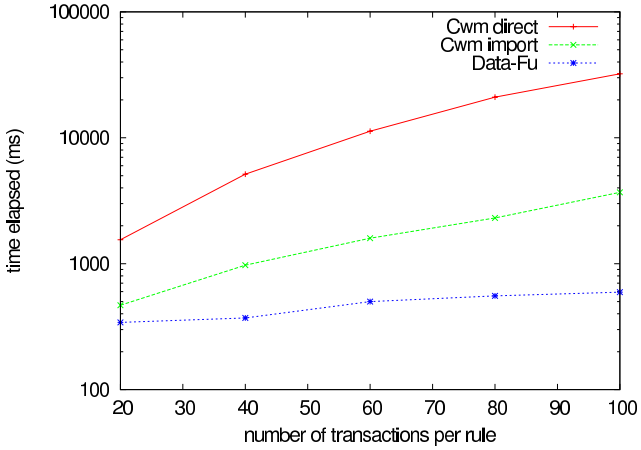


Figure 5: Average execution time from ten runs for different evaluation set-ups with one rule

The Data-Fu engine is able to execute the interaction by orders of magnitude faster than the other two approaches with Cwm. Also the growth-rate of the execution time with the increasing size of number sets is much lower with Data-Fu compared to the Cwm approaches (note the log scale in Figure 5). The Data-Fu engine achieves this time saving by leveraging the data flow network: Data-Fu has just to put the new results after an interaction through the data flow network to find new bindings. Cwm on the other hand has to apply the rules repeatedly over the increasing dataset in its triple store.

To evaluate the capabilities of the Data-Fu engine with regard to parallelisation we run the same interaction of retrieving successors of numbers again, with ten different "kinds" of numbers (A-J) in parallel. The numbers are distinguished by different namespaces. Each of the three evaluation set-ups requires ten rules for the interaction (each addressing another namespace), analog to the previously shown rules. Figure 6 illustrates this evaluation set-up.

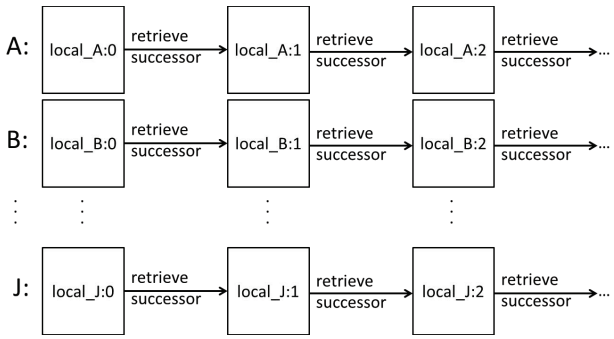


Figure 6: Interactions of evaluation set-up with one rule

The results for the different evaluation set-ups are shown in Table 4 and Figure 7 as average from ten runs. Again Data-Fu executes the interaction significantly faster with a lower growth rate than Cwm in the other set-ups: In the case of the most interactions (10 x 100) *Cwm direct* requires over 17 minutes and *Cwm import* over 32 seconds, the Data-Fu engine handles the same interactions in under 4 seconds.

Table 4: Average execution time from ten runs for different evaluation set-ups with ten rules in parallel

number set size	Data-Fu	Cwm direct	Cwm import
20	1833 ms	22513 ms	2836 ms
40	2421 ms	108421 ms	7067 ms
60	2916 ms	310498 ms	13518 ms
80	3889 ms	621798 ms	21729 ms
100	3944 ms	1038524 ms	32983 ms

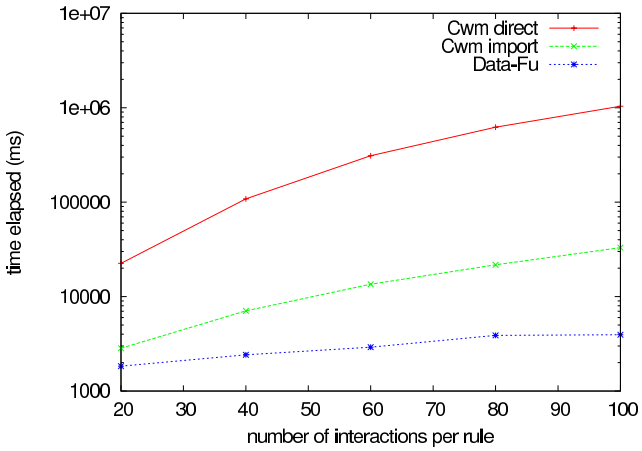


Figure 7: Average execution time from ten runs for different evaluation set-ups with ten rules in parallel

Comparing the results of the interactions with a single rule and the interactions with ten rules in parallel we note, that the Data-Fu engine suffers less than Cwm from the ten times increased workload when executing ten rules in parallel. On average for the individual sizes of number sets

- *Data-Fu* requires 6.2 times longer,
- *Cwm direct* requires 25 times longer,
- *Cwm import* requires 8 times longer,

when running with ten rules compared to one single rule.

The reason for this time advantage is the capability of the Data-Fu engine to execute several components of the interaction in parallel, e.g., the evaluation of the triple patterns of the queries and the communication with several web resources. Note, that the theoretically possible speedup due to parallelisation on a dual core system implies that a 10 times increased workload results in a 5 times longer execution time. However, the Data-Fu engine cannot quite reach this optimal speedup, since not all parts in the interaction can be completely parallelised, e.g., the management of the individual threads. These parts of an interaction that cannot be completely parallelised result in a slightly diminished speedup, as stated by Amdahl's Law [3].

Following the results of the evaluation in comparison with Cwm, we devise a final evaluation setting to test the scalability of the Data-Fu engine when performing large amounts of interactions. Similar to the previous evaluation setting we retrieve number resources that are identified during runtime as successor of an already found number. We fix the size of the number sets to 100, i.e., we deploy sets of 100 consecutive number resources that are distinguished with their namespace. Then we retrieve the numbers of every set with a respective rule. We evaluate the runtime of the Data-Fu engine with 20, 40, 60, 80 and 100 rules/number sets, thus performing between 2 000 and 10 000 interactions. Additionally we measure the time needed to calculate the evaluation plan separately to compare it with the total execution time. The results are shown in Table 5 and Figure 8.

Table 5: Average execution time from ten runs of Data-Fu engine with number sets of size 100

rules/number sets	execution time	evaluation plan
20	8357 ms	4 ms
40	17195 ms	6 ms
60	30767 ms	7 ms
80	49430 ms	8 ms
100	75764 ms	9 ms

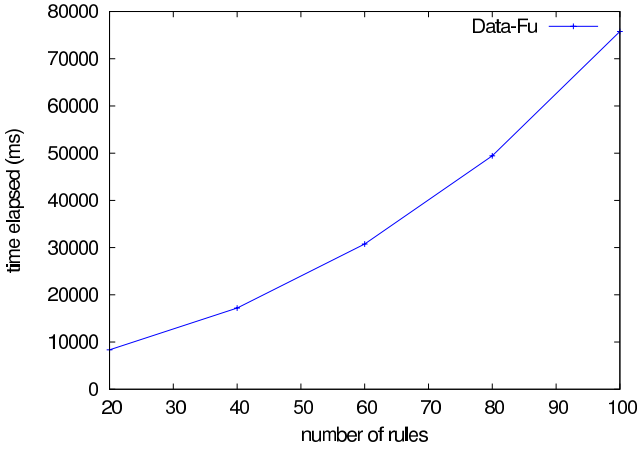


Figure 8: Average execution time from ten runs of Data-Fu engine with number sets of size 100

The results of the evaluation for large amounts of interactions show that the Data-Fu engine scales well up to thousands of interactions even on commodity hardware. The Data-Fu engine is capable of interacting with 10 000 web resources in about 1:15 min. The necessary time required to establish the evaluation plan increases with the number of rules, but remains a very small fraction of the overall execution time and is therefore negligible.

The evaluation shows the advantages of the parallel processing of queries and interactions and provides evidence that the Data-Fu engine is capable of performing rapid interactions with web resources as desired. We did not consider the necessary time to establish HTTP connections on the web and the response time of the servers, where resources are deployed, since these additional time requirements would be the same for any employed interaction system. Note however, that due to its parallel processing nature, the Data-Fu engine could further benefit from longer response times of servers compared to other systems: At the same time as the Data-Fu engine performs the manipulations and retrieval of resources other rules can be evaluated, thus the overall execution time can be minimised.

We provide the data used for the evaluation and an executable jar online[12] to re-run the experiments.

[12] `http://people.aifb.kit.edu/sts/datafu/evaluation/`

8. RELATED WORK

Pautasso introduces an extension to BPEL [21] for a composition of REST and traditional web services. REST services are wrapped in WSDL descriptions to allow for a BPEL composition. Our approach focuses on a native composition of REST services, rather than relying on technologies of traditional web services. For a comparison between RESTful services and "big" services see [23].

There exist several approaches that extend the WS-* stack with semantic capabilities by leveraging ontologies and rule-based descriptions (e.g., [28, 10, 8]) to achieve an increased degree of automation in high level tasks, such as service discovery, composition and mediation. Those approaches extending WS-* became known as Semantic Web Services (SWS). An Approach to combine RESTful services with SWS technologies in particular WSMO-Lite [31] was investigated by Kopecky et al. [16]. In contrast to SWS, REST architectures do not allow to define arbitrary functions, but are constrained to a defined set of methods and are built around another kind of abstraction: the resource. Therefore our approach is more focused on resource/data centric scenarios in distributed environments (e.g., in the Web).

Active XML introduces service calls as XML nodes that are placeholders for new XML documents that can be retrieved from the service [1]. The service calls are comparable to hypermedia links in resource descriptions and the active XML document corresponds to the knowledge space. In contrast to Active XML, our work discovers links to new resources instead of links to function calls. The resource model provides more flexibility, e.g., a Data-Fu program could perform a DELETE on a discovered resource, whereas the Active XML equivalent would be constrained to the predefined operations in the original link.

The scripting language S [6] allows to develop Web resources with a focus on performance due to parallelisation of calculations. Resources can make use of other resources in descriptions, thus also enabling a way of composing REST services. S does not explicitly address the flexibility of REST and has no explicit facilities to leverage hypermedia controls or to infer required operations from resource states.

RESTdesc [30] is an approach in which RESTful Linked Data resources are described in N3-Notation. The composition of resources is based on an N3 reasoner and stipulates manual interventions of users to decide which hypermedia controls should be followed.

Hernandez et al. [14] proposes a model for semantically enabled REST services as a combination of pi-calculus [19] and approaches to triple space computing [9] pioneered by the Linda system [13]. They argue, that the resource states can be seen as triple spaces, where during an interaction triple spaces can be created and destroyed as proposed in an extension of triple space computing by Simperl et al. [25]. Our service model is in contrast to this approach more focused on the composition of data driven interactions.

Similar to the idea of triple spaces is the composition of RESTful resources in a process space, proposed by Krummenacher et al. [17] based on resources described using graph patterns. Speiser and Harth [26] propose similar descriptions for RESTful Linked Data Services. Our approach shares the idea that graph pattern described resources read input from and write output to a shared space. We improve on this approach by providing a service model and a more explicit way of defining the interaction with resources.

9. CONCLUSION

In this paper, we addressed the problem of creating value-added compositions of data and functionalities. As a unifying model for both static data sources and dynamic services, we described how Linked Data Resources can be extended with descriptions for RESTful manipulation. The natural extension of Linked Data with RESTful manipulation of resources enables a framework with uniform semantic resource representations for REST architectures. We have proposed to exploit the advantages resulting from the combination of REST and Linked Data in a programming framework for the Semantic Web. We have introduced Data-Fu, a declarative rule-based execution language with a state transition system as formal grounding, and the challenges we address with this language, i.e., achieving scalability and performance while preserving the flexibility and robustness of REST. Furthermore, we described our implementation of an execution engine for the Data-Fu language.

For future work, we plan to extend our approach in the following directions. First, we will add capabilities to improve handling of failures of resource interactions. Second, we will extend our formal model of Data-Fu to provide clearly defined semantics in the presence of non-deterministic rules. Third, we will integrate support for rule-based reasoning into the execution engine. The rules bring useful expressivity for aligning different vocabularies and can be easily supported in the engine by introducing triple-producing rule heads in addition to the current state transition handlers.

Acknowledgments

This work was partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641) and by the German Ministry of Education and Research (BMBF) within the SoftwareCampus project framework.

10. REFERENCES

[1] S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *Proceedings of the 23rd Symposium on Principles of Database Systems (PODS'04)*, pages 35–45. ACM, 2004.

[2] A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. *SIGMOD Record*, 21(2):59–68, 1992.

[3] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the 1967 Spring Joint Computer Conference (AFIPS'67)*, pages 483–485, Atlantic City, New Jersey, 1967. ACM.

[4] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: A programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, pages 119–130, Indianapolis, Indiana, USA, 2010. ACM.

[5] T. Berners-Lee. *Read-Write Linked Data*. August 2009. Avaiable at `http://www.w3.org/DesignIssues/ReadWriteLinkedData.html`, accessed 26th November 2012.

[6] D. Bonetta, A. Peternier, C. Pautasso, and W. Binder. S: A scripting language for high-performance RESTful

web services. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12)*, 2012.

[7] C. Brenner, A. Fensel, D. Fensel, A. Gagiu, I. Larizgoitia, B. Leiter, I. Stavrakantonakis, and A. Thalhammer. How to domesticate the multi-channel communication monster. Available at `http://oc.sti2.at/sites/default/files/oc_short_handouts.pdf`.

[8] J. Cardoso and A. Sheth. *Semantic Web Services, Processes and Applications*. Springer, 2006.

[9] D. Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In *Proceedings of the IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'04)*, number 3283 in Lecture Notes in Computer Science, pages 43–53, Bangkok, Thailand, 2004. Springer.

[10] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2006.

[11] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[12] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[13] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7:80–112, 1985.

[14] A. G. Hernández and M. N. M. García. A formal definition of RESTful semantic web services. In *Proceedings of the First International Workshop on RESTful Design (WS-REST'10)*, pages 39–45, 2010.

[15] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys'07)*, pages 59–72, Lisbon, Portugal, 2007. ACM.

[16] J. Kopecky, T. Vitvar, and D. Fensel. MicroWSMO: Semantic description of RESTful services. Technical report, WSMO Working Group, 2008.

[17] R. Krummenacher, B. Norton, and A. Marte. Towards Linked Open Services. In *Proceedings of the 3rd Future Internet Symposium (FIS'10)*, volume 6369 of *Lecture Notes in Computer Science*, Berlin, Germany, 2010. Springer.

[18] E. A. Lee and P. Varaiya. *Structure and Interpretation of Signals and Systems*. Addison-Wesley, 2011.

[19] R. Milner. *Communicating and Mobile Systems: π-calculus*. Cambridge University Press, Cambridge, UK, 1999.

[20] B. Norton and S. Stadtmüller. Scalable discovery of linked services. In *Proceedings of the 4th International Workshop on REsource Discovery (RED'11)*, 2011.

[21] C. Pautasso. RESTful web service composition with BPEL for REST. *Journal of Data and Knowledge Engineering*, 68(9):851–866, 2009.

[22] C. Pautasso and E. Wilde. Why is the web loosely coupled?: A multi-faceted metric for service design. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 911–920, Madrid, Spain, 2009. ACM.

[23] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[24] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2007.

[25] E. Simperl, R. Krummenacher, and L. Nixon. A coordination model for triplespace computing. In *Proceedings of the 9th International Conference on Coordination Models and Languages (COORDINATION'07)*, 2007.

[26] S. Speiser and A. Harth. Integrating Linked Data and services with Linked Data Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC'11) Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 170–184, Heraklion, Crete, Greece, 2011. Springer.

[27] S. Stadtmüller and A. Harth. Towards data-driven programming for RESTful Linked Data. In *Workshop on Programming the Semantic Web (ISWC'12)*, 2012.

[28] R. Studer, S. Grimm, and Abecker, A. (eds.). *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007.

[29] M. Taheriyan, C. A. Knoblock, P. A. Szekely, and J. L. Ambite. Rapidly integrating services into the Linked Data cloud. In *Proceedings of the 11th International Semantic Web Conference (ISWC'12)*, volume 7649 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2012.

[30] R. Verborgh, T. Steiner, D. V. Deursen, R. V. de Walle, and J. G. Valls. Efficient runtime service discovery and consumption with hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices (NWeSP'11)*, Salamanca, Spain, 2011.

[31] T. Vitvar, J. Kopecky, M. Zaremba, and D. Fensel. WSMO-Lite: Lightweight semantic descriptions for services on the web. In *Proceedings on the 5th European Conference on Web Services (ECOWS'07)*, pages 77–86, 2007.

[32] J. Webber. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly, 2010.

[33] M. Weiss and G. R. Gangadharan. Modeling the mashup ecosystem: Structure and growth. *R&D Management*, 40(1):40–49, 2010.

[34] E. Wilde. REST and RDF granularity, 2009. Available at `http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html`.

[35] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems (PDIS'91)*, pages 68–77, Miami Beach, FL, USA, 1991. IEEE Computer Society Press.

[P6] Andreas Harth and Steffen Stadtmüller. Parallel Processing of Rule-based Programs on Linked Data. 2015. Under review.

Parallel Processing of Rule-based Programs on Linked Data

Andreas Harth
Institute AIFB, Karlsruhe Institute of Technology (KIT)
Englerstr. 11, 76131 Karlsruhe, Germany
Email: harth@kit.edu

Steffen Stadtmüller
Institute AIFB, Karlsruhe Institute of Technology (KIT)
Englerstr. 11, 76131 Karlsruhe, Germany
Email: steffen.stadtmueller@kit.edu

***Abstract*—Linked Data provides popular means for decentralised data publishing on the web, with HTTP as access protocol and RDF as data model. Linked Data enables a new class of link-traversal algorithms that intertwine the evaluation of query plans with network requests to fetch data. That is, the dataset over which queries are carried out is not fixed, but iteratively expanded during query processing. We join two currently isolated strands of research to provide query processing capabilities over web sources: (i) methods for evaluating queries over interlinked sources via link traversal; and (ii) approaches for integrating data over interlinked schemas via reasoning. Instead of using distinct systems for each task, we propose a unified three-stage architecture, with a data-driven evaluation model for rule-based programs, to allows for parallel streaming processing of cyclic (recursive) pipelined query plans in conjunction with network requests. We design and benchmark various algorithms that implement the architecture, and show that a streaming approach outperforms a batching approach. Our execution model allows users to individually manage distinct thread pools for I/O-bound and CPU-bound tasks, and thus to balance both types of tasks.**

I. Introduction

The Linked Data principles offer a uniform data representation format and access mechanism for data on the web [1], where an estimated 30 billion statements are attainable [1,2]. In large distributed environments such as the web, the integration of multiple data sources provides value. A minimal set of conventions facilitates integration. In the case of Linked Data, data format and access protocol are fixed, and mappings between sources via hyperlinks lead to a very large interconnected data graph. Data providers use constructs from RDFS and OWL to help clients interpret and align the schema and data, often between different sources.

Consequently, an application that uses Linked Data needs to implement several steps: the data has to be accessed and downloaded, the data has to be integrated, and the integrated data has to be queried. To access, integrate and query Linked Data, one has to use multiple specialised systems in combination: a crawler for collecting data from distributed resources; a reasoner to resolve differences in modelling and schema; and an RDF store [2], [3] to provide query processing capabilities. The last two steps may be combined using systems such as OWLim [4], Oracle's Semantic Data Store [5], RDFox [6] that provide query answering and reasoning in a combined system. Still, with these systems, accessing data is a separate step. Consequently, applications that require access to frequently updated sources suffer from suboptimal performance. Alternatively, one has to implement the data processing and retrieval in imperative programs (as opposed to declarative programs); those mash-ups are often tailored for a fixed number of data sources in a narrowly-defined domain.

Example 1: Consider the following query: return publications authored by programme committee members of a given conference. Such a query might be interesting for exploring related work relevant for a scientific community. Data about PC members may come from the conference website, and data about publications may come from bibliographic databases such as DBLP, but also national libraries such as the Library of Congress. Increasingly, such data is available as Linked Data. To get an exhaustive set of results, it is not sufficient to go to a single site. Rather, given the linkage between resources, we can follow links to other resources describing the person or publication, align the retrieved data, and evaluate the query over the combination of data. In case there are updates to sources, we can re-run the process and return updated results.

We design and study methods to access and integrate data from distributed but interlinked sources. In our approach, the different steps required to access and integrate data can be encoded in a high-level specification based on rules. We describe the architecture of a system that is able to follow links for data retrieval, integrate the data via reasoning over retrieved schemata, and evaluate queries, all in a cohesive integrated process. The set of data sources (and schema axioms) is not fixed a priori, but is incrementally expanded during runtime. We provide a forward-chaining mechanism for evaluating arbitrary rulesets. Our method is able to use rulesets that partially encode the semantics of RDFS and OWL, with restrictions regarding the handling of datatypes and an infinite amount of axiomatic triples (as in RDFS) [7], [8]. We evaluate our approach with an end-to-end system implementing the described architecture.

Our rule formalism amounts to a variant of datalog/production rules (the equivalent of RIF Core[3], a minimal formalism that encompasses both deduction and production rules). The basic problems in datalog are computationally hard: fact entailment is EXPTIME-complete in combined complexity, and PTIME-complete in data complexity. The latter implies that the problem of rule evaluation is inherently serial under standard complexity-theoretic assumptions [7]. Nevertheless

[1] http://www4.wiwiss.fu-berlin.de/lodcloud/state/
[2] http://wp.sigmod.org/?p=786
[3] http://www.w3.org/TR/rif-core/

we achieve significant speedup in practice on the Lehigh University Benchmark (LUBM).

In particular, we address the following challenges:

- To retrieve all available resource (or a very large subset, e.g., every resource in a specific domain) is prohibitively expensive. Applications cannot follow all available links blindly, but should be designed in a manner to target specified links. Further, there is little coordination between providers, and resources have to be aligned and integrated. However, schema information for data integration has to be acquired at runtime.
- The retrieval of data can be time consuming, due to network latency and limited available bandwidth. Data access can stall the actual processing and query evaluation; thus, data access has to be decoupled from data processing. In addition, a system capable of handling sizable amounts of data has to be multi-threaded. Further, we operate on the web, and as such we focus on data access using (polling-based) HTTP GET, as opposed to stream processing systems which rely on an event-based model for data access [9], [10], [11], [12].

Several link traversal systems exist that evaluate queries directly over sources accessible as Linked Data [13], [14], [15], [16]. However, these systems do not take the semantics of data sources (the mappings of schema and instance elements) into account during query processing. Other systems implementing reasoning [17], [6] typically operate over locally accessible single-source datasets; we assume a hyperlinked environment where data access and data processing are interleaved. Dataspace systems [18] rely on a centralised catalogue of sources; on the web, we assume hyperlinks between sources for resource discovery. Stream reasoning systems [19] and complex event processing systems [20] rely on a fixed number of sources that push data. On the web, polling is the prevalent communication mode. Further, in an environment based on resources and polling, we are able to discover new sources and new data (including new reasoning constructs) at runtime.

Our contributions are as follows:

- We describe rule-based programs that encode reasoning features intertwined with link traversal specifications. In particular, we introduce the notion of request rules, which infer required network lookups from the processed data. Request rules complement deduction rules, which infer and materialise implicit information from the processed data. We define syntax and semantics for such rule-based programs (Section III).
- We introduce a generic architecture for the parallel evaluation of queries and rules with cyclic operator plans, and input/output components that are decoupled from data processing. Our generic architecture can be instantiated with multiple threading models, both using a batch model which proceeds in rounds, and a streaming model where each data item is processed immediately. Further we describe a push-based scheduling execution model which goes beyond traditional pipelined models [21] or scheduling models [22]. Thus the model avoids overhead from inter-process communication and at the same time caters to scenarios with network requests, where data from multiple sources has to be processed on arrival. Specifically, our model separates the workload related to network access from the workload related to local data processing. Thus, we can balance available computing resources between data processing and network lookups to minimise overall runtime (Section IV).

The techniques described in this paper are also applicable to distributed query processing and stream reasoning systems. While we benchmark different threading models on shared memory architectures, the methods we use come from the relational database toolbox, and can be implemented in other shared-nothing architectures such as Apache Hadoop MapReduce (batch model) and Apache Spark Streaming (streaming model).

We introduce definitions and the problem statement in Section II, introduce syntax and semantics of rule-based programs in Section III, give an overview of the system and explain the architecture in Section IV, describe experiments to determine the optimal balance for CPU and I/O-bound work for each threading model in Section V, present related work in Section VI and conclude with Section VII.

II. Preliminaries

We now define the necessary concepts of RDF and graph patterns as foundation for rule-based programs. We stay as close to existing definitions [23], [24] as possible. For a comprehensive treatment of RDF see [25].

The basis of Linked Data is RDF, which consists of triples:

Definition 1: (RDF Term, RDF Triple, RDF Graph) Let $\mathcal{U}$, $\mathcal{B}$, $\mathcal{L}$ be pairwise disjoint infinite sets representing respectively URIs, blank nodes and literals. Let $\mathcal{T}$ be the set of all RDF terms $\mathcal{T} = \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. An RDF triple is defined as $\langle s, p, o \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \times \mathcal{B} \times \mathcal{L})$. An RDF Graph is a finite set of RDF triples $G \subset (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \times \mathcal{B} \times \mathcal{L})$.

Graph-structured RDF data may come from multiple RDF files:

Definition 2: (RDF Dataset) An RDF dataset D is a set $\{G_0, (u_1, G_1), \ldots, (u_n, G_n)\}$ where $G_0 \ldots G_n$ are RDF graphs and $u_1 \ldots u_n$ are distinct URIs. The graph G_0 is called default graph, and $G_1 \ldots G_n$ are called named graphs. For a set of RDF Graphs Γ, $\bigcup_{G \in \Gamma} G$ is the merged RDF graph, i.e., the union of all triples $\langle s, p, o \rangle \in G,\ \forall G \in \Gamma$ while forcing any shared blank nodes that occur in more than one graph to be distinct[4].

Linked Data mandates that a HTTP `GET` request on a resource (identified via an URI) returns an RDF graph:

Definition 3: (Request) We write $G = httpget(u)$ for the RDF graph returned for an HTTP `GET` request on the URI $u \in \mathcal{U}$ of a resource.

Please note that a server might redirect a request to a URI u to another URI u' before an RDF graph is returned, due to the correspondence between abstract resources and information resources (see [23] for details).

[4]For the definition of RDF merge see http://www.w3.org/TR/rdf11-mt/#shared-blank-nodes-unions-and-merges.

III. Rule-based Programs

We now formally introduce a language to specify rule-based programs. We define two kinds of rules: request rules, to support the specification of link traversal, and deduction rules, to support the integration of heterogeneous data. We further provide an operational semantics for such rule-based programs, followed by an example.

A. Syntax

We use Notation3 as syntax for programs. Notation3 extends standard RDF syntax with (universally quantified) variables and graph quoting to express relationships between graphs using curly brackets [26]. We first introduce variables, next graph quoting, and finally some built-in URIs to specify rules.

To be able to specify queries and rules, we require variables, and the concept of a basic graph pattern (BGP).

Definition 4: (Triple Pattern, Basic Graph Pattern) Let $\mathcal{V}$ be the infinite set of variables, disjoint with $\mathcal{U}$, $\mathcal{B}$ and $\mathcal{L}$. A triple pattern TP is a triple $\langle s, p, o\rangle \in (\mathcal{V} \cup \mathcal{U} \cup \mathcal{B}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, where subject s, predicate p and object o can either be a variable or an RDF Term. A basic graph pattern (BGP) is a set of triple patterns $Q = \{t_1, ..., t_n\}$.

For queries, we support standard SPARQL syntax, but focus on a subset of SPARQL that supports BGP queries[5]. BGPs form the basis of the `WHERE` clause in SPARQL queries; we support `SELECT` and `CONSTRUCT` result clauses.

A `SELECT` query includes a list of variables that should be returned:

`SELECT` $variables$
`WHERE` { BGP }

For `CONSTRUCT` queries we introduce the notion of graph template.

Definition 5: (Graph Template) A graph template consists of a set of triples from $(\mathcal{T} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{T} \cup \mathcal{V})$.

A `CONSTRUCT` query includes a graph template that specifies how variables from BGP should be used to form new triples:

`CONSTRUCT` { $graph\ template$ }
`WHERE` { BGP }

We use BGPs also as basis for both deduction and request rules. To encode rules, we use the Notation3 implication (=>, short for `log:implies`).

A deduction rule has the following form:

{ BGP } => { $graph\ template$ } .

We call the subject of a => triple the body (or antecedent), and the object of a => triple the head (or consequent) of the rule. In other word, the body of a deduction rule consists of BGP, while the head consists of a $graph\ template$.

[5]Additional SPARQL constructs can be layered upon BGP queries.

To be able to specify request rules, we introduce the notion of request template.

Definition 6: (Request Template) A request template consists of a HTTP method (currently only `GET`) and a request target $\mathcal{U} \cup \mathcal{V}$.

As syntax for request templates, we use URIs from the `http` and `httpm` vocabularies. We support the `http:mthd` to specify the HTTP method (currently only `httpm:GET`), and the `http:requestURI` property for the request target. Requests are written as triples, with a blank node on subject position. Please note that we ignore blank nodes in request templates for now. In follow-up work, we could define a semantics that instantiates the blank node with identifiers for actual requests that are carried out during program evaluation.

The following encodes a HTTP GET request to the URI `dcc:complete`[6]:

```
[] http:mthd httpm:GET ;
   http:requestURI dcc:complete .
```

Requests with a fixed request target can be specified as ground RDF triples. Requests with both fixed and variable request targets can appear as the head of a request rule. A request rule has the following form:

{ BGP } => { $request\ template$ } .

We impose syntactic restrictions on rules. Rules have to be safe, that is, a variable that appears in the rule head also has to appear in the rule body. Further, we do not allow blank nodes in graph templates in deduction rules[7]. We do, however, allow literals on subject position of graph template triples (so-called generalised RDF [8]). Those generalised triples are used during the rule evaluation, and filtered out before serialising triples into an RDF syntax.

B. Semantics

We now describe the meaning of rules. For a comprehensive theoretical study of SPARQL CONSTRUCT queries including recursion see [27], [28].

In the following, we write $terms(S)$ for the set of RDF terms in S, where S can be a triple, graph, triple pattern, graph pattern, graph template or request template, and $vars(S)$ for the set of variables in S.

Definition 7: (Solution Mapping, Instance Mapping, Solution Sequence) Let μ be a partial function that maps variables to RDF terms: $\mu : \mathcal{V} \to \mathcal{T}$. Let $dom(\mu) \subset \mathcal{V}$ denote the domain of μ, i.e. the subset of $\mathcal{V}$ where μ is defined. An RDF instance mapping is a partially defined function $\sigma : \mathcal{B} \to \mathcal{T}$ that maps blank nodes to arbitrary RDF terms.

Let Q be a BGP and G an RDF graph. Further let the pattern instance mapping $P^{\sigma}_{\mu} : vars(Q) \cup terms(Q) \to \mathcal{T}$ be

[6]Assuming the prefix dcc set to http://data.semanticweb.org/conference/dc/2010/.

[7]We exclude blank nodes from rule heads to guarantee the existence of a fixpoint.

a function that maps variables and RDF terms in Q to RDF terms[8]:

$$P_\mu^\sigma(x) = \begin{cases} \mu(x) & \text{if } x \in \mathcal{V} \\ \sigma(x) & \text{if } x \in \mathcal{B} \\ x & \text{if } x \in \mathcal{U} \cup \mathcal{L} \end{cases}$$

Let $P_\mu^\sigma(vars(Q) \cup terms(Q))$ denote the RDF graph resulting from a substitution of all elements $e \in vars(Q) \cup terms(Q)$ in the BGP Q according to $P_\mu^\sigma(e)$. A mapping μ for the variables $vars(Q)$ is a solution mapping for Q from G if $P_\mu^\sigma(vars(Q) \cup terms(Q))$ is a subgraph of G; i.e., μ satisfies that $\exists\sigma\forall\langle s,p,o\rangle \in Q : \langle P(s), P(p), P(o)\rangle \in G$ and $dom(\mu) = vars(Q)$.

A solution sequence is possibly unordered list of solution mappings. We write $\Omega_G(Q)$ for the set[9] of all unique solution sequences for the BGP Q from RDF graph G, i.e.,

$$\Omega_G(Q) = \{\mu |\ \exists\sigma :\ P_\mu^\sigma(vars(Q) \cup terms(Q)) \subseteq G\}$$

Solution mappings and sequences are part of the standard definition of SPARQL semantics.

Definition 8: (Request Rule) An execution step of a request rule ρ^r with head H and body B over a graph G adds the graph G_{resp} to G, where G_{resp} is an RDF graph returned by HTTP requests. The resources for the requests are identified by the URIs $\mu(x) \in \mathcal{U}$, to which x is mapped in all solution sequences of B from G, i.e., $\mu(x) \in \Omega_G(B)$

Therefore the graph G_{resp} is defined as

$$G_{resp} = \begin{cases} \bigcup_{\mu\in\Omega_G(B)} httpget(\mu(x)) & \text{if } x \in \mathcal{V} \wedge \mu(x) \in \mathcal{U} \\ \emptyset & \text{otherwise} \end{cases}$$

We denote f^r as the function that maps a request rule ρ^r and a graph G to the graph of retrieved triples resulting from one execution step of the request rule:

$$f^r(\rho^r, G) = G_{resp}$$

Request rules allow developers to determine in a fine grained manner what resources to retrieve and which links to follow. The resources are identified by the solution sequence for the variable provided in the rule head. For every identified solution sequence of the rule body from the processed graph, the provided variable signifies a resource to lookup. If the mapping for the identified variable of a solution sequence points to a URI, the URI is the identifier of a resource to retrieve. In other words, our evaluation procedure determines URIs from the processed data and thus dynamically follows and expands links between resources.

Similar to deduction rules, request rules are applied recursively to a graph, where the retrieved triples are added to the graph after every step. The recursive application is necessary, because the retrieved data in one step can lead to the identification of further links to follow.

[8]We deviate slightly from the SPARQL specification in http://www.w3.org/TR/sparql11-query/ by explicitly defining pattern instance mappings as function over RDF terms and variables. However, the semantics remains unchanged.

[9]Please note that we assume set semantics for simplicity, in-line with [29], [30].

Definition 9: (Deduction Rule) An execution step of a deduction rule ρ^d with head H and body B over a graph G adds the graph G_{add} to G, where G_{add} is the RDF graph resulting from the substitution of the variables in H according to all solution sequences of B from graph G, i.e., $\forall\mu \in \Omega_G(B)\forall\langle s,p,o\rangle \in H : \langle P_\mu^\sigma(s), P_\mu^\sigma(p), P_\mu^\sigma(o)\rangle \in G_{add}$ with a unique blank node mapping σ for every $\mu \in \Omega_G(B)$. We denote f^d as the function that maps a deduction rule ρ^d and a graph G to the graph of derived triples G_{add} resulting from one execution step of the deduction rule:

$$f^d(\rho^d, G) = G_{add}$$

Deduction rules can be used to encode application-specific reasoning constructs. Languages such as RDFS or OWL provide modelling primitives such for expression subclass relationships between classes or domain/range restriction of properties. On the Linked Data web, many sources reuse class and property URIs that are described using RDFS or OWL constructs. To support certain constructs commonly used for schema- or instance-level mappings, a general-purpose rule engine such as the one we describe can make use of these mappings. Rulesets that encode different entailment regimes are readily available on the web, e.g., the OWL LD ruleset [31], which covers OWL constructs that are widely used in practice.

We combine deduction rules and request rules to form *programs*:

Definition 10: (Program) A program $\mathcal{P} = (G, R, \mathrm{P}^d, \mathrm{P}^r)$ is a a tuple with:

- G a finite set of initial triples, i.e., the starting graph;
- R a finite set of request templates with fixed target;
- P^d a finite set of deduction rules; and
- P^r a finite set of request rules.

There has to be at least either one triple in the starting graph or one initial request specified, that is, $G \neq \emptyset \vee R \neq \emptyset$.

The execution of a program implies that the initial requests are carried out and the returned triples are added to the starting graph G. The rules P^d and P^r are recursively executed over G until a fixpoint is reached, i.e., no additional triples can be derived and no additional requests can be carried out. We denote st as the function mapping to the resulting graph of an execution step of multiple deduction and request rules over a graph G:

$$st(\mathrm{P}^d, \mathrm{P}^r, G) = G \cup \bigcup_{\rho^d\in\mathrm{P}^d} f^d(\rho^d, G) \cup \bigcup_{\rho^r\in\mathrm{P}^r} f^r(\rho^r, G)$$

The complete execution of a program $\mathcal{P}$ is the recursive evaluation of n execution steps of all rules $\mathrm{P}^d \in \mathcal{P}$ and $\mathrm{P}^r \in \mathcal{P}$ until a fixpoint is reached:

$$min(n) : (st_1 \circ ... \circ st_n)(\mathrm{P}^d, \mathrm{P}^r, G) = (st_1 \circ ... \circ st_{n+1})(\mathrm{P}^d, \mathrm{P}^r, G)$$

We denote $G_\mathcal{P}$ as the graph resulting from the execution of the program $\mathcal{P}$. We denote $R_\mathcal{P}$ as the set of URIs of all resources retrieved by a program, which includes resources from initial requests as well as resources retrieved via request rules.

BGP queries can be registered to programs. If a BGP query Q is registered to a program $\mathcal{P}$, the query Q is evaluated over

the result graph $G_{\mathcal{P}}$ of program $\mathcal{P}$. We denote $\Omega_{\mathcal{P}}(Q)$ as the solution sequence of query Q from program $\mathcal{P}$.

The evaluation of a program continuously extends the initial graph G. Specifically, both deduction and request rules monotonically add triples to G and are not capable of removing triples. Neither deduction rules nor request rules can generate new RDF terms, i.e., terms that are not present in the initial graph G, the URIs $R_{\mathcal{P}}$ of retrieved resources, or the rules P^d and P^r. Consequently, given that $R_{\mathcal{P}}$ is finite, a program is guaranteed to have a fixpoint, as the number of combinations of RDF terms to form valid triples is finite. Therefore, eventually a program will not be able to infer new triples in another recursion to add to the result graph, and reach the fixpoint. Consequently, the result graph will also be a finite set of triples.

However, on the web, resources might be dynamically generated[10], which can cause a program to attempt to iteratively retrieve a very large or even infinite $R_{\mathcal{P}}$. Thus, the program cannot reach a fixpoint. There may be other cases where we do not want to wait for a program to reach its fixpoint (see Section IV for alternative termination criteria for programs).

C. Example Program Evaluation

In the following we describe in detail the scenario from the running example. To get a list of publications authored by PC members of a given conference, say, we can use a program $\mathcal{P}$ with an empty initial graph $G = \emptyset$, and an request to a resource representing the conference we are interested in, i.e., `dcc:complete` for the Dublin Core 2010 conference.

Consider the following program $\mathcal{P}_{ex}$:

```
# (0) Initial request
[] http:mthd httpm:GET ;
   http:requestURI dcc:complete .

# (1) Request rule to retrieve information about PC members
{ dcc:programme-committee-member swc:heldBy ?pcmember .
} => {
  [] http:mthd httpm:GET ; http:requestURI ?pcmember .
} .

# (2) Request rule to retrieve information about
# publications of PC members.
{ dcc:programme-committee-member swc:heldBy ?pcmember .
  ?pcmember foaf:made ?publication .
} => {
  [] http:mthd httpm:GET ; http:requestURI ?publication .
} .

# (3) Deduction rule to specify inverse relationship
# between foaf:made and foaf:maker
{ ?x foaf:made ?y . } => { ?y foaf:maker ?x . } .
```

A system evaluating $\mathcal{P}_{ex}$ first retrieves the initial resource (0), which includes triples for the first triple pattern of the query (URIs of PC members). Request rule (1) specifies that all `swc:heldBy` links to RDF representations of PC members are followed. Request rule (2) specifies that all `foaf:made` links to RDF representations of publications of PC members are followed. Deduction rule (3) specifies that `foaf:made` and `foaf:maker` are the inverse of each other.

[10]As example consider the set of resources representing natural numbers (c.f. http://km.aifb.kit.edu/projects/numbers/) and a program that always retrieves the successor for every found number.

Additionally we register the following SPARQL `SELECT` query to $\mathcal{P}_{ex}$ that returns the names of the PC members and the titles of the publications they authored:

```
SELECT ?name ?title
WHERE {
 dcc:programme-committee-member swc:heldBy ?pcmember .
 ?pcmember foaf:name ?name .
 ?pcmember foaf:made ?publication .
 ?publication dcterms:title ?title .
}
```

A user might want to include predefined rulesets (such as OWL LD [31]) to provide for certain OWL entailments. With the OWL LD rules, the program evaluation would take `owl:sameAs` statements into account, and thus retrieve URIs to PC members and publications that are connected to the initial URIs of both via `owl:sameAs`.

IV. Methods and Algorithms

We begin the system description with a problem statement in Section IV-A. Then, we introduce the architecture in Section IV-B, consisting of three stages (Input, Processing, Output). Next, we explain the execution model in Section IV-C, followed by detailed descriptions of each stage. Finally, we cover termination in Section IV-F.

A. Problem Statement

We assume a set of distributed heterogeneous Linked Data sources, each identified by a URI $d \in D$. Users want to evaluate SPARQL BGP queries over the RDF graph G_D resulting from the retrieval of all sources in D, i.e., $G_D = \bigcup_{d \in D} httpget(d)$. However, only a subset of the URIs of the resources are known before the evaluation. The remaining URIs have to be discovered at runtime via link traversal.

We provide the means to specify which links to follow and how to interpret the meaning of vocabularies in the form of rule-based programs. Therefore the input for the considered problem consists of a program $\mathcal{P}$ and a set of SPARQL BGP queries Q that are to be evaluated over the retrieved data. Please note that the user specifies both $\mathcal{P}$ and Q.

The program evaluation includes query and rule processing in tandem with data access. The output are the results of the queries executed over the retrieved data.

The optimisation goals are to increase throughput (i.e., process as many triples and sources as quickly as possible) and decrease latency (i.e., return results as quickly as possible) in the face of the combination of data access and processing. The point about latency is related to "anytime" behaviour, a trade-off between processing time and query results. We study an "online query processing" algorithm where results improve gradually with increased processing time [32]. Such a setup is also relevant for stream reasoning systems [33].

Query and rule processing tasks can be considered CPU-bound, i.e., the core on which the tasks are executed are almost completely utilised and the computing power of the core determine throughput. Data access tasks can be considered I/O-bound, i.e., only a fraction of the core on which the tasks are executed is utilised, as network latency and bandwidth determine throughput. Consequently we have to execute programs

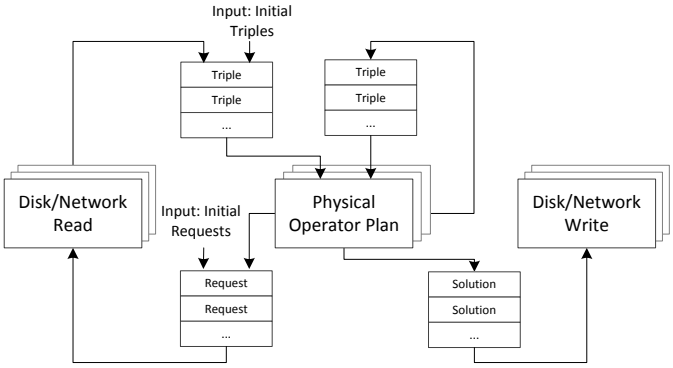


Fig. 1. Illustration of the data flow between the three stages: Input, Processing, Output. We use queues to transfer items between the stages. The BaseTripleQueue takes as input initial triples, and the InputQueue initial requests. The Disk/Network Read component reads RDF from disk and network, puts the resulting triples into the BaseTripleQueue, and receives new requests from the Physical Operator Plan. The Physical Operator Plan operates on triples from the BaseTripleQueue and DerivedTripleQueue, and generates solutions that are passed via the SolutionQueue to the Disk/Network Write component. The Physical Operator Plan also generates new triples, which are fed back to itself via the DerivedTripleQueue.

in a way to accommodate for the heterogeneous workloads of data processing and access.

B. Architecture

In this section we describe an architecture for a system that evaluates programs. We propose a system architecture that consists of three stages: Input, Processing and Output, similar to SEDA (staged event-driven architecture) [34]. The Input and Output stages are responsible for reading and writing data, respectively (Section IV-D). The Processing stage consists of a physical operator plan for identifying result bindings for the BGPs in rule bodies and queries, and for generating triples and requests from templates (Section IV-E). Figure 1 illustrates the dataflow between stages.

The physical operator plan is derived from P^d, P^r and Q in three steps known from relational database implementation:

1) Parse the program $\mathcal{P}$ and the queries Q, and build an internal representation of P^d, P^r and Q. Remember initial triples G and initial requests R.
2) Create the logical operator plan from the internal representation of P^d, P^r and Q. Please note that the logical operator plan contains cycles to encode the recursive nature of P^d.
3) Create the physical operator plan encoded with the equivalent of unnamed relational algebra expressions.

The actual evaluation starts with G and R as input to the physical operator plan. We use queues for the passing of work items between the stages:

- InputQueue: for requests to be performed by the Disk/Network Read component, filled by the Physical Operator Plan.
- BaseTripleQueue: for triples from requests to be processed by the Physical Operator Plan, filled by the Disk/Network Read component.
- SolutionQueue: for solutions to queries to be handled by the Disk/Network Write component.

Please note that data flows between Disk/Network Read and Physical Operator Plan in both directions: both stages mutually influence each other. The queues allow for balancing the CPU-bound and the I/O-bound tasks: with the appropriate queue configuration, the Physical Operator Plan is able to exert back-pressure to the Disk/Network Read component. That is, the architecture caters for flow control between stages.

We further use a queue within the Physical Operator Plan to break the cycle due to recursion:

- DerivedTripleQueue: for triples from derivations generated by the Physical Operator Plan.and to be processed by the Physical Operator Plan again.

We need the DerivedTripleQueue to avoid arbitrary deep call stacks, which would occur with a direct feedback loop within the Physical Operator Plan due to the recursion. Please note that we eliminate duplicates at the DerivedTripleQueue to be able to reach the fixpoint.

To sum up: output to queries are routed to the Disk/Network Write component via the SolutionQueue. Output from derivation rules are connected to the DerivedTripleQueue. Output from request rules are routed to the Disk/Network Read component via the InputQueue.

The different stages work independently of each other, emitting data or processing streams while performing I/O-bound or CPU-bound tasks along the way. The stages operate until the system determines that the program evaluation is finished. Criteria for termination can include:

- a specified timeout has been reached;
- the system has requested a pre-defined number of sources;
- the system has reached a pre-defined depth in traversing the graph of sources, starting from the initial sources; or
- the computation has reached a fixpoint, i.e., the result graph $G_{\mathcal{P}}$ is completely calculated according to the rules of the program and all query solutions have been completely generated and written.

In the remainder of the paper we consider reaching the fixpoint as termination condition. However, additional termination conditions can be used to prevent a program from running indefinitely, e.g. if the program cannot reach a fixpoint due to an infinite amount of resources provided by a "spider trap". In these cases, the anytime behaviour of the system helps, as the system outputs results as soon as they are generated.

Our proposed architecture satisfies the following characteristics:

- The processing has to happen in parallel to benefit from multiple cores. An important requirement is the ability to balance I/O-bound and CPU-bound work.
- The system generates a stream from the resources that are polled, and processes the arriving data in a streaming fashion. The query processing works in streaming fashion to be able to return results incrementally.
- Data processing and I/O can be interleaved to reduce overall elapsed time relative to stepwise evaluation. Interleaving work requires incremental processing, because new requests are derived from the processed data, and new

requests have to be done as soon as possible to account for latency in network sources.
- The processing of rules and queries has to be optimised to avoid unnecessary work. We describe a set of standard database query optimisation techniques that can be used in the architecture in Section V.

We assume three distinct thread pools, one for each stage:

- InputWorker: take requests from the InputQueue, carry out requests, and parse data into the BaseTripleQueue.
- ProcessingWorker: take triples from the BaseTripleQueue and DerivedTripleQueue, evaluate queries and rules, and put results into either the InputQueue, DerivedTripleQueue, or SolutionQueue.
- SinkWorker: take solution sequences from the SolutionQueue and write them to disk or network.

In the following, we first describe how to workers operate on data. We then cover the Input and Output stages, followed by the Processing stage. Finally, we present methods for detecting termination in different configurations of the architecture with regards to multi-threading.

C. Push-Based Scheduling Execution Model

In the following we detail our push-based scheduling execution model for the parallel evaluation of linked programs. For parallel evaluation of programs, we do not use an individual thread for every operator in the physical operator plan as proposed in [16], [21]. Instead we allow operators to schedule each other within a single thread, i.e., every operator iterates over its intermediate results and pushes the results to subsequent operators in the plan with a process call. Therefore we avoid the overhead of inter-process communication [22], which would result from passing intermediate results between threads.

Traditional iterator execution models [22], [13] are demand-driven, i.e., operators request data items for processing from preceding operators. In contrast we propose a data-driven push-based scheduling of operators, where the operators immediately push intermediate results to subsequent operators. Such a push-based execution model especially caters to data processing scenarios that include network lookups, as data can be processed immediately when it arrives, rather than waiting on operators during slow network request.

Every ProcessingWorker thread evaluates a sequence of scheduled operators and the number of employed ProcessingWorker threads determines the number of in parallel evaluated operator sequences. Consequently, the ProcessingWorker operate on the shared physical operator plan. The set of InputWorker perform the derived network lookups and feed retrieved data back to the operator plan via the BaseTripleQueue.

ProcessingWorker threads are CPU-bound, i.e., in general a running ProcessingWorker thread utilises almost completely the core on which the thread is executed. Therefore the number of ProcessingWorker is driven by the number of available cores: fewer threads than available cores would not fully utilise the capacity of the system and more threads can not generate a benefit, as the existing threads are already able to fully utilise the system. In fact, early experiments with a 1:1 relation between operators and threads showed an excessive amount of operating system context switches.

InputWorker threads are I/O-bound, i.e., a running InputWorker thread utilises only a fraction of the core on which the thread is executed, as requests are constrained by network bandwidth and latency. The number of InputWorker threads can therefore exceed the number of available cores until the overhead of coordinating the threads outweighs the benefit of using parallel requests.

SinkWorker threads are also I/O-bound, as serialising results is inexpensive. For simplicity, we assume one SinkWorker per registered query.

D. Disk/Network Read and Write

The goal of the Disk/Network Read component is to carry out requests as specified in request rules, and parse the responses; requests are incrementally generated by the physical operator plan out of identified result bindings for request rules. Thus, the component actually performs the link traversal, as it carries out requests on resources whose URIs are extracted from processed graphs. Although we have implemented also support for `file` URIs, we assume in the following only `http` URIs.

The goal of the Disk/Network Write component is to serialise query results, and write the results to disk or network. There are as many SinkWorker threads as there are queries. There is one queue for each SinkWorker. As the implementation of the Disk/Network Write component is straightforward, we focus in the following on the Disk/Network Read component.

Intuitively, the Disk/Network Read component operates similar to a web crawler: the component performs requests in parallel while spacing out requests to avoid overloading sources, and ensures that each request is only done once.

The Disk/Network Read component has to satisfy the following requirements:

- Retrieve data as fast as possible, to not become the bottleneck in processing.
- Do not repeat requests.
- Carry out requests in a polite fashion, i.e., do not overload servers with parallel requests[11]

Input to the Disk/Network Read component via the InputQueue are URIs of resources to retrieve $\mu(x)$, which are derived from the result bindings for a request rule body:

$$\mu(x) \in \mathcal{U} \text{ with } \mu \in \Omega_G(B) \text{ for } \rho^r : \{B\} \implies \{x\}$$

The requests on the resources in the InputQueue are carried out in parallel by the InputWorker threads. Output of the Disk/Network Read component are the RDF graphs returned from retrieving the resources, $httpget(\mu(x))$. The triples of the returned RDF graphs $t \in httpget(\mu(x)) \subseteq G_{resp}$ are fed into the BaseTripleQueue so that the triples can be further processed. Thus, the request rules also establish cycles, which reflects the recursive application of the request rules.

[11] It used to be customary to wait several seconds between requests to the same server. Within our architecture we can ensure delays with a modified InputQueue.

TABLE I. OPERATORS USED IN THE LOGICAL AND PHYSICAL PLANS TO PROCESS DATA.

Operator	Description
Input	Distinguished node for input of triples
TriplePattern	Evaluate triple patterns
EquiJoin	Compute equi-join between two inputs
Binding	Store intermediate results for EquiJoins
Project	Project tuples for `SELECT` queries
Construct	Generate new triples for `CONSTRUCT` queries
Consume	Output query results
Derivation	Generate derived triples
Request	Generate requests

If multiple solutions of a request rule body map the variable in the rule head to the same URI, the same request is derived multiple times.

$$\exists \mu_1, \mu_2 \in \Omega_G(B) : \mu_1(x) = \mu_2(x) \text{ for } \rho^r : \{B\} \implies \{x\}$$

As duplicate requests to the same resource cause unnecessary source server load and network traffic, the Disk/Network Read component has to ensure to issue only unique requests. Thus, the Disk/Network Read component can maintain a set of all URIs that were already used for requests in a visited set. Multiple requests on the same URI are then rejected. Multiple requests on the same URI might also occur if a previously unknown URI is requested that simply redirects to an already seen URI. Consequently the system also needs to store information about redirects, i.e., which URIs redirected to the information resource that eventually delivered data. When performing a large number of requests across many web servers, there are many more things to consider, such as politeness and dead host detection, which we omit due to space constraints.

E. Physical Operator Plan

The physical operator plan is a network of operators (see Table I for a list of operators). The BGPs of rule bodies and queries are converted into expressions involving TriplePattern, Binding and EquiJoin.

The network calculates the result bindings of the rule bodies and queries from data items sent through the operator plan.

- Results $\Omega_G(B)$ for the body B of a deduction rule ρ^d are used to generate derivations via Derivation.
- Results $\Omega_G(B)$ for the body B of a request rule ρ^r are used to generate requests via Request.
- Results $\Omega_G(Q)$ of a query Q are the final output, sent to Consume via Project for `SELECT` and Construct for `CONSTRUCT` queries.

The operators are connected via sender/receiver relationships. The workers propagate solution sequences through the physical operator plan along the sender/receiver relationships. The solution sequences do not carry full variable/term pairs, but are represented as tuples of RDF terms, i.e., unnamed relational algebra expressions. The position of an RDF term in the tuple determines which variable maps to the term.

Individual triple patterns of BGPs of queries and rule bodies are represented with TriplePattern operators. BGPs with multiple triple patterns lead to multiple TriplePattern operators, connected via EquiJoin operators, which represent join conditions. Thus, each BGP is represented as a tree and the combination of all BGPs in the program and queries form a forest of join trees. However, different trees of the join tree forest can share operators to eliminate common subexpressions and thus duplication of work and space. Thus the forest forms a connected graph, rather then a set of independent join trees.

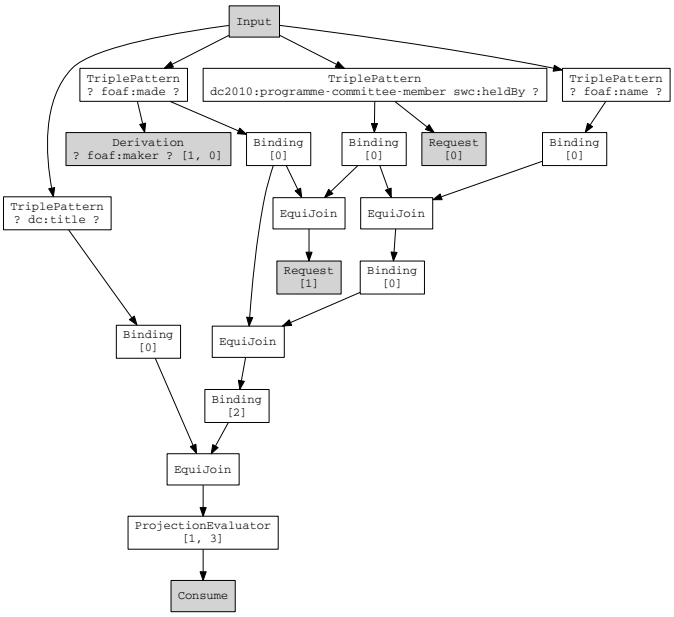


Fig. 2. Physical operator plan for rules and query of the example, after common subexpression elimination. Grey boxes indicate connections to the queues: the Input operator is polling from the BaseTripleQueue, the Derivation operator adds derived triples to the DerivedTripleQueue, and the Request operators add new requests to the InputQueue. Finally, the Consume operator places query solutions into the SolutionQueue for serialisation and output.

The Input operator is the distinguished node in the operator plan, the place where to input data, which connects to all TriplePattern operators. The trees receive tuples via the TriplePattern operators, which in turn receive triples from the Input operator. Depending on whether the BGP is from a query, a deduction rule or a request rule, the top operator of the BGP tree is connected to an Output operator (via Project operators), to a Derivation operator, or to a Request operator.

Instead of a directed acyclic graph of operators, which is the basis for most dataflow-systems, our operator graph may contain cycles. The possible cycles established by connections from the Derivation operators to the Input operator, i.e, the cycles reflect the recursive application of deduction rules. That is, the Derivation operators send the generated triples $t \in G_{add}$ back to the Input operator for processing. We break cycles in the physical operator plan with the DerivedTripleQueue between Derivation operators and Input operator.

Derived requests are sent to the InputQueue, which feeds the network request component. Thus, the physical operator plan and request component are decoupled, which allows to take into account the different processing speeds of different worker threads, i.e., balance I/O-bound and CPU-bound tasks.

We have implemented and evaluated the following optimisations:

- Reordering heuristics based on number of variables in triple patterns.

- Common subexpression elimination to avoid duplication of work and waste of space. To be able to collapse a join tree, we store the hash tables of the symmetric hash join in separate operators (Binding operators), so that multiple EquiJoin operators can access the same hash table containing intermediate results[12].

Figure 2 illustrates the physical operator plan for deduction rules, request rules and query of the example. Please note that the figure shows the plan after common subexpression elimination: several Binding operators have multiple receiving EquiJoin operators.

F. Termination

Termination is a challenging topic in our architecture, given the cyclic dataflow plans. To determine when a program has been completely evaluated (i.e., all implied lookups are completed and all inferences from the rules are drawn) we have to identify the fixpoint of the program evaluation. The fixpoint has been reached when two conditions are met:

- All queues are empty, i.e., there are no further triples or requests to process.
- All worker threads are idle or sleeping, i.e., the worker threads are not processing triples or requests.

In a multi-threaded architecture, detecting that these conditions hold can be easy or quite involved, depending on how the parallelism is instantiated. In the following, we describe different instantiations and ways to check for termination in each.

Serial: With the serial threading model, there is only one thread, taking the roles of InputWorker, ProcessingWorker and SinkWorker. There is no parallel processing, and all operations are carried out sequentially in the main thread. The main thread takes one item from the queue, processes the item, and pushes the results to the queue. First, all items in the DerivedTripleQueue are processed, followed by items from the BaseTripleQueue, then items from the InputQueue, and finally items from the SolutionQueuel When the thread fails to poll a new item from any of the queues (that is, all queues are empty), the fixpoint has been reached. We use the single-threaded *serial* model as baseline.

Batching: A straightforward approach to a multi-threaded model is to process a program in rounds [35]: in round n, derived triples and requests are stored into new queues for the next round $n+1$. Once the queues in round n are empty, round $n+1$ starts. The fixpoint is reached once the queues are empty at the beginning of a new round. The *batching* model does not fully leverage the available system resources towards the end of each round: as a queue contains fewer items than there are threads, only some of the threads are processing the last items while the other threads are sleeping, even though the queues for round $n+1$ might already contain items.

Streaming: With the *streaming* threading model, derived triples and requests are directly pushed into the queues. If a worker is idle but finds its queue to be empty, the worker sleeps and periodically wakes to try to acquire an item from the queue again (busy waiting[13]). The wait time can be freely chosen and results in a trade off: A short waiting time can cause system overhead as workers wake up to often unnecessarily; a long waiting time can cause the workers to stay unnecessarily asleep while items in the queue are present. To identify the fixpoint the main thread of control checks periodically whether i) the queues are empty and ii) all workers are in sleeping state.

V. Experiments

We now describe our experiments as part of a systematic evaluation. We analyse the behaviour of a fully implemented system for the parallel evaluation of rule-based programs. In particular, we experiment with different threading models and analyse the system in terms of throughput with different degrees of parallelism, i.e., a different number of worker threads and cores.

A. LUBM-LD Benchmark

The widely-used Lehigh University Benchmark (LUBM) [36] provides queries over a synthetically generated dataset from the university domain. Our experiments are based on LUBM, adapted to adhere to the Linked Data principles.

The LUBM dataset generator creates separate files containing instances of `univ-bench:University` and `univ-bench:Department`. In addition, the generated files include links from `univ-bench:University` instances to associated instances of `univ-bench:Department`. However, there is no correspondence between the identifiers of entities (e.g., `http://www.Department7.University19.edu/`) and files (e.g., `University19_7.owl`).

We adapted LUBM so that URIs for the classes `univ-bench:University` and `univ-bench:Department` correspond to the URIs of the files that contain the RDF describing each. In addition, we added an index file that contains references to instances of `univ-bench:University`, to be able to use link traversal for data access. We also changed the serialisation syntax to Turtle, as XML does not allow for relative URIs in namespaces[14]. Using URI references (relative URIs) is important, so that the directory with data can be moved to different locations while ensuring link consistency.

We use LUBM-LD 100 for all our experiments. LUBM-LD 100 consists of 2.108 files (plus one file with the vocabulary), leading to a total size of 807 MB in Turtle syntax[15] with 13.892.172 triples. We host the LUBM-LD files on a local server which is configured with a network delay of 50 ms, to provide network delays similar to the ones on the web. We do not cap the bandwidth, and we do not assume access restriction to the server. In other words, we evaluate the system with fully parallel access[16].

[12] In Rete production system terminology, the Binding operators after TriplePattern could be called α-memory, and the Binding operators after EquiJoin operators β-memory.

[13] In our experiments, a blocking implementation based on barriers turned out to be slower than the busy waiting implementation.

[14] http://www.w3.org/2000/09/xppa

[15] The amount of transferred data via HTTP is slightly higher than the file size due to additional header data.

[16] The system can adhere to basic politeness criteria based on a different implementation of the InputQueue that avoids parallel requests to the same host and optionally limits the access rate.

B. Setup

We run experiments with a machine with two Intel Xeon E5-2670 2.60GHz processors (8 physical cores per processor; hyper-threading results in 32 logical cores) and 256 GB of main memory. The prototype system implementing the architecture is written in Java, and the experiments are run on OpenJDK 1.7u79[17]. We use lighthttpd on a separate machine to serve the LUBM files. In our test the load of the web server machine never exceeded 0.05. Both machines are connected via 1Gbit/s Ethernet and run Debian GNU/Linux Wheezy 64bit.

C. Workload

For the experiments we use the following program that specifies an initial request to the index file, and then iteratively performs requests on instance URIs of instances of `univ-bench:University` and `univ-bench:Department`. We include the OWL LD ruleset [31].

```
# Initial request: get index file
[] http:mthd httpm:GET ;
   http:requestURI <./Universities.ttl> .

# Request rule to retrieve information about University
{ ?x a ub:University . }
=>
{ [] http:mthd httpm:GET ;
       http:requestURI ?x . } .

# Request rule to retrieve information about Department
{ ?x a ub:Department . }
=>
{ [] http:mthd httpm:GET ;
       http:requestURI ?x . } .

# Request rule to retrieve file with T-Box
{ ?x owl:imports ?y . }
=>
{ [] http:mthd httpm:GET ;
       http:requestURI ?y . } .
```

In addition to the program, we register the 14 LUBM queries; query results are serialised to disk in TSV format.

D. Baselines

Given our novel scenario, there is no system available to match our functionality. We however give baseline performance results which help to put our results in context. Please be aware, however, that the baseline measurements of related systems are not directly comparable, as none of the systems allows for link traversal. The baseline systems assume that all links are available at the beginning, which is not the case in the link traversal scenario, where loading data and processing data mutually influence each other. The baseline systems concern the Input and Processing/Output stages.

As baseline for Input, we use the `rapper` utility from Dave Beckett's Redland RDF libraries[18]. Downloading and parsing the LUBM-LD 100 files using GNU `parallel` with 32 parallel jobs takes 111.08s. As baseline for Processing/Output, we use Ontotext's GraphDB Lite (formerly OWLIM), version 6.1.8410. We run LUBM 100 with the standard configuration of the supplied benchmark script, but enabled larger Java heap size (`-Xmx200G`). Loading, materialisation and query processing takes 306.01s. Thus, accessing data, processing data and outputting results takes 417.09s (111.08s+306.01s).

E. Results and Analysis

We now report results for the *serial*, *batching* and *streaming* threading models. We implemented the *streaming* model both as a blocking model based on barriers (which is the standard in the textbooks and libraries) and using busy spinning. Surprisingly, the busy spinning model outperforms the blocking model based on barriers. We thus include only results for the busy spinning model in the remainder under *streaming*.

We first report results for *serial*: 2929.17s without any optimisations, 2935.12s for just reordering, 1162.68s with both reordering and common subexpression elimination.

Figure 3 shows the results for different thread configurations for the *batching* and *streaming* variants[19]. *Streaming* is almost twice as fast as *batching*. The optimal configuration for *batching* is 32 InputWorker threads and 32 ProcessingWorker threads, for *streaming* the optimum is 64 InputWorker threads and 24 ProcessingWorker threads. The insight that too many processing threads are detrimental to performance rules out approaches that use a 1:1 mapping between operators and threads, given that our rule programs often yield operator plans with many more operators than cores.

We also determine the effect of reordering and common subexpression elimination for these optimal configurations. For *batching* without optimisations 2061.17s, 2019.00s with reordering and 236.49s with both reordering and common subexpression elimination. For *streaming* without optimisations 1270.09s, 1326.04s with reordering and 142.88s with both reordering and common subexpression elimination.

The fact that reordering sometimes is even worse than just using the order as written in the BGPs of the rules and queries indicates that the explicitly stated BGP order in OWL LD rules and LUBM queries is efficient.

In summary we can say that the presented architecture combining I/O-bound and CPU-bound work outperforms systems where both stages are considered separately. *Streaming* outperforms *batching*, and common subexpression elimination turns out to have a significant effect on overall performance, given that there is a large overlap of patterns in rules in the OWL-LD ruleset.

VI. Related Work

There are many approaches for link-traversal query processing, first described in [13]. Ladwig et al. [16] propose a push-based model for query processing over Linked Data similar to ours. However, they implement their system using the Scala actor model, with a queue and a lightweight thread for each operator, and do not consider rules. Umbrich et al. [24]

[17] We use the default Garbage Collector settings with `-Xmx200G -Xms200G`.

[18] http://librdf.org/

[19] We carried out experiments with half the reported step size, but only show partial results due to space constraints.

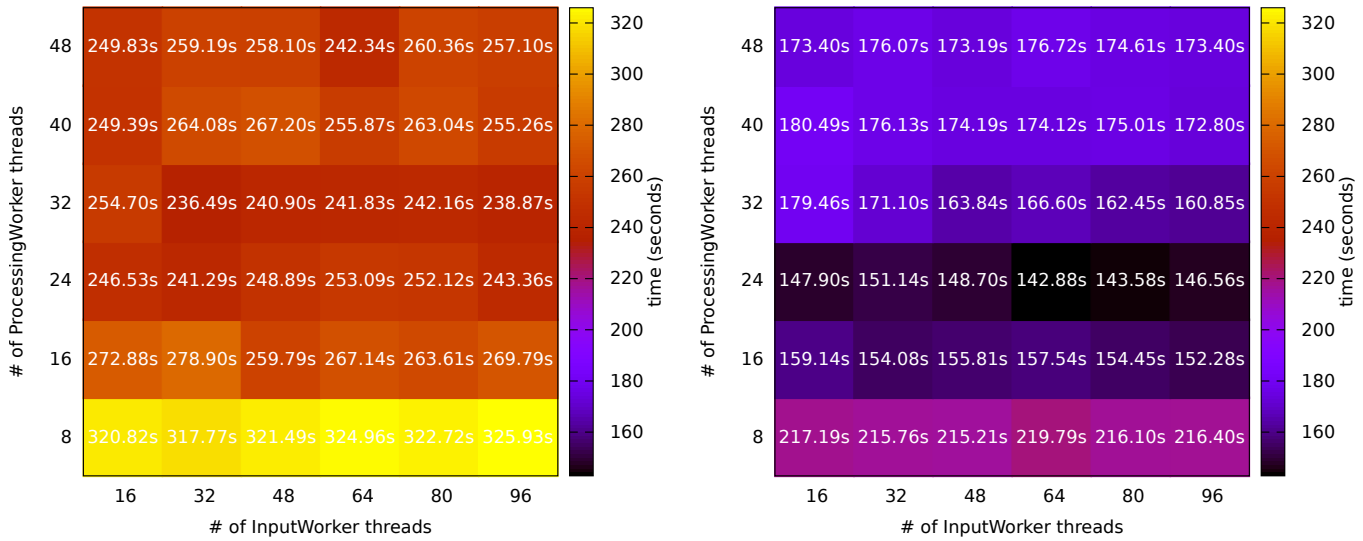


Fig. 3. Elapsed time (lower is better) for data access, processing (reasoning and query evaluation), and output of query results for LUBM-LD 100. The left graph shows results for the *batching* model, and the right graph shows results for the *streaming* model.

study the impact on the recall of query results when taking into account reasoning constructs. In contrast, we describe methods to carry out rule-based reasoning in combination with HTTP requests. In contrast to [15] who specify path expressions on queries, we only allow for processing of BGP queries. However, we can specify link expansion in a more fine-grained manner using request rules.

Stadtmueller et al. [37] propose a system similar to ours that also takes into account data manipulation (CRUD) operations. In contrast, we focus on read operations and on the parallel execution, specify a concrete syntax for request rules, describe several optimisations, and conduct an extensive performance evaluation.

Grosof et al. [38] note that rules can be rewritten in the face of T-Box statements, yielding more efficient rule sets. The drawback for using the T-Box to optimise rule processing is that the T-Box needs to be known (and fixed) before evaluating the query plan. Given that we aim for a general-purpose link-following, in which the system might access hitherto unknown T-Box statement during query evaluation, the optimisation does not apply in our scenario.

Many approaches rely on an a priori fixed T-Box. A strand of work in rule-based reasoning revolves around the idea of separating T-Box (triples with `rdf:type` property and those with RDF, RDFS and OWL vocabularies) and A-Box. Hogan et al. [39] were the first to use the fact that many rulesets consist of joins between T-Box statements and A-Box statements. Their system holds T-Box in memory, and does scans over A-Box triples to perform the join. Similarly, [40] uses specialised algorithms tailored to a particular ruleset. In contrast, we provide a general rule processor for positive Datalog on triples, that allows e.g., for arbitrary join operations over instance data. Motik et al. [6] also propose a parallel system for general recursive Datalog rules, where multiple processes extract facts from a database and evaluate the fact over sub-queries obtained from the rules. The result of the evaluation is written back to the database and consequently available for the evaluation of other sub-queries. The approach of Motik et la. relies on the availability of a storage scheme that allows for efficient evaluation of the sub-queries (i.e., indexes over the the facts in the database) and efficient update mechanisms. In our approach, the data is fetched from the network, and triples are directly pushed through the operators which maintain multimaps for joins.

We use a variant of the the symmetric hash join operator as proposed by Wilschut and Apers [21]. To enable parallel processing Wilschut describes a pipeline execution model, where every operator is executed by a process. In our pipeline model data is "pushed" from one operator to another by inter-process communication (i.e., pipes). Further, in our version, the hash tables are external to the join operation (the Binding operator), to allow for reusing hashtables with bindings across joins. Graefe [22] argues that the overhead introduced by the interprocess communication can be avoided by employing an execution model that allows operators to schedule each other within a single process. In Graefe's model operators can request new data items from other operators whenever new data items are required. The operators iterate over the received (pull) data items to produce new (intermediate) results, and thus eliminate the need to buffer data in pipelines. However, the author points out that in a scenario where data-sources have to unload data as it arrives, a more data-driven push-based model might be more appropriate. We use such a push-based model.

VII. Conclusion

We have presented methods for query processing over interlinked data sources based on declarative programs specified using rules. The specifications include request rules, to take into account the links between sources to be able to discover new resources in a decentralised environment, and deduction rules, to specify the semantics of data items as defined in knowledge representation languages such as RDFS and fragments of OWL.

We have described and evaluated several threading models for parallel evaluation of rules and queries in a streaming fashion based on a data-driven push-based execution model. We believe that the presented formalism and approach will make it easier for users and applications to consume the wealth of structured data available as Linked Data on the web.

REFERENCES

[1] T. Berners-Lee, "Linked Data," Cambridge, Massachusetts, USA, 2006.

[2] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: Sextuple indexing for semantic web data management," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 1008–1019, Aug. 2008.

[3] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold, "Column-store support for rdf data management: Not all swans are white," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1553–1563, Aug. 2008.

[4] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "Owlim: A family of scalable semantic repositories," *Semantic Web Journal*, vol. 2, no. 1, pp. 33–42, Jan. 2011.

[5] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan, "Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle," in *Proc. of the 24th International Conference on Data Engineering*, 2008, pp. 1239–1248.

[6] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu, "Parallel materialisation of datalog programs in centralised, main-memory RDF systems," in *Proc. of the 28th AAAI Conference on Artificial Intelligence*, 2014, pp. 129–137.

[7] P. F. Patel-Schneider, "Reasoning in RDFS is inherently serial, at least in the worst case," in *Proc. of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, November 11-15, 2012*, 2012.

[8] H. J. ter Horst, "Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary," *Web Semant.*, vol. 3, no. 2-3, pp. 79–115, Oct. 2005.

[9] S. D. Viglas, J. F. Naughton, and J. Burger, "Maximizing the output rate of multi-way join queries over streaming information sources," in *Proc. of the 29th International Conference on Very Large Data Bases - Volume 29*. VLDB Endowment, 2003, pp. 285–296.

[10] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker, "Operator scheduling in a data stream manager," in *Proc. of the 29th International Conference on Very Large Data Bases*, 2003, pp. 838–849.

[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing," in *Proc. of the 22nd International Conference on Management of Data*, 2003, pp. 668–668.

[12] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik, "Retrospective on aurora," *The VLDB Journal*, vol. 13, no. 4, pp. 370–383, Dec. 2004.

[13] O. Hartig, C. Bizer, and J.-C. Freytag, "Executing SPARQL queries over the web of linked data," in *Proc. of the 8th International Semantic Web Conference*, 2009.

[14] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich, "Data summaries for on-demand queries over linked data," in *Proc. of the 19th International Conference on WWW*. ACM, 2010, pp. 411–420.

[15] V. Fionda, C. Gutierrez, and G. Pirró, "Semantic navigation on the web of data: Specification of routes, web fragments and actions," in *Proc. of the 21st International Conference on WWW*, 2012, pp. 281–290.

[16] G. Ladwig and T. Tran, "Linked data query processing strategies," in *Proc. of the 9th International Semantic Web Conference*, 2010, pp. 453–469.

[17] M. Sintek and S. Decker, "TRIPLE - A query, inference, and transformation language for the semantic web," in *Proc. of 1st International Semantic Web Conference*, 2002, pp. 364–378.

[18] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Record*, vol. 34, pp. 27–33, Dec 2005.

[19] A. Margara, J. Urbani, F. van Harmelen, and H. Bal, "Streaming the web: Reasoning over dynamic data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 25, no. 0, 2014.

[20] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, "Stream reasoning and complex event processing in ETALIS," *Semantic Web*, vol. 3, no. 4, pp. 397–407, 2012.

[21] A. N. Wilschut and P. M. G. Apers, "Dataflow query execution in a parallel main-memory environment," in *Proc. of the 1st International Conference on Parallel and Distributed Information Systems*, 1991, pp. 68–77.

[22] G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Surveys*, vol. 25, no. 2, pp. 73–169, Jun. 1993.

[23] A. Harth and S. Speiser, "On Completeness Classes for Query Evaluation on Linked Data," in *Proc. of the 26th National Conference on Artificial Intelligence*, 2012.

[24] J. Umbrich, A. Hogan, A. Polleres, and S. Decker, "Improving the recall of live linked data querying through reasoning," in *Proc. of the 6th International Conference on Web Reasoning and Rule Systems*, 2012, pp. 188–204.

[25] C. Gutierrez, C. Hurtado, and A. O. Mendelzon, "Foundations of semantic web databases," in *Proc. of the 23rd Symposium on Principles of Database Systems*, 2004, pp. 95–106.

[26] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax," W3C, W3C Team Submission, Mar. 2011.

[27] E. V. Kostylev, J. L. Reutter, and M. Ugarte, "CONSTRUCT Queries in SPARQL," in *Proc. of the 18th International Conference on Database Theory*, vol. 31, 2015, pp. 212–229.

[28] J. L. Reutter, A. Soto, and D. Vrgoc, "Recursion in SPARQL," in *Proc. of the 14th International Semantic Web Conference*, October 2015, pp. 19–35.

[29] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," *ACM Transactions on Database Systems*, vol. 34, pp. 16:1–16:45, September 2009.

[30] M. Schmidt, M. Meier, and G. Lausen, "Foundations of sparql query optimization," in *Proc. of the 13th International Conference on Database Theory*, 2010, pp. 4–33.

[31] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres, "Owl: Yet to arrive on the web of data?" *CoRR*, vol. abs/1202.0984, 2012.

[32] P. J. Haas and J. M. Hellerstein, "Online query processing: A tutorial," in *Proc. of the 19th International Conference on Management of Data*. ACM, 2001, pp. 623–.

[33] E. D. Valle, S. Ceri, F. v. Harmelen, and D. Fensel, "It's a streaming world! reasoning upon rapidly changing information," *IEEE Intelligent Systems*, vol. 24, no. 6, pp. 83–89, Nov. 2009.

[34] M. Welsh, D. Culler, and E. Brewer, "Seda: An architecture for well-conditioned, scalable internet services," in *Proc. of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 230–243.

[35] F. N. Afrati and J. D. Ullman, "Transitive closure and recursive datalog implemented on clusters," in *Proc. of the 15th International Conference on Extending Database Technology*, 2012, pp. 132–143.

[36] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *J. Web Sem.*, vol. 3, no. 2-3, pp. 158–182, 2005.

[37] S. Stadtmüller, S. Speiser, A. Harth, and R. Studer, "Data-fu: A language and an interpreter for interaction with read/write linked data," in *Proc. of the 22nd International Conference on WWW*, 2013, pp. 1225–1236.

[38] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: Combining logic programs with description logic," in *Proc. of the 12th International Conference on WWW*, 2003, pp. 48–57.

[39] A. Hogan, A. Harth, and A. Polleres, "Saor: Authoritative reasoning for the web," in *Proc. of the Sixth International Semantic Web Conference, Third Asian Semantic Web Conference*, Dec 2008, pp. 76–90.

[40] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal, "OWL reasoning with webpie: Calculating the closure of 100 billion triples," in *Proc. of the 7th Extended Semantic Web Conference*, 2010, pp. 213–227.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Habilitationsleistung selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Karlsruhe, den 21.12.2015