Link Traversal with Distributed Subweb Specifications

Bogaerts, Bart; Ketsman, Bas; Zeboudj, Younes; Aamer, Heba; Taelman, Ruben; Verborgh, Ruben

Published in:
International Joint Conference on Rules and Reasoning

DOI:
10.1007/978-3-030-91167-6_5

Publication date:
2021

Document Version:
Accepted author manuscript

Link Traversal with Distributed Subweb Specifications

Bart Bogaerts[1], Bas Ketsman[1], Younes Zeboudj[1], Heba Aamer[2], Ruben Taelman[3], and Ruben Verborgh[3]

[1] Vrije Universiteit Brussel, Belgium — {firstname.lastnamei}@vub.be
[2] Universiteit Hasselt, Hasselt, Belgium — heba.mohamed@uhasselt.be
[3] Ghent University – imec – IDLab — {firstname.lastname}@ugent.be

Abstract. Link Traversal–based Query Processing (LTQP), in which a SPARQL query is evaluated over a web of documents rather than a single dataset, is often seen as a theoretically interesting yet impractical technique. However, in a time where the hypercentralization of data has increasingly come under scrutiny, a decentralized Web of Data with a simple document-based interface is appealing, as it enables data publishers to control their data and access rights. While LTQP allows evaluating complex queries over such webs, it suffers from performance issues (due to the high number of documents containing data) as well as information quality concerns (due to the many sources providing such documents). In existing LTQP approaches, the burden of finding sources to query is entirely in the hands of the *data consumer*. In this paper, we argue that to solve these issues, *data publishers* should also be able to suggest sources of interest and *guide* the data consumer towards relevant and trustworthy data. We introduce a theoretical framework that enables such guided link traversal and study its properties. We illustrate with a theoretic example that this can improve query results and reduce the number of network requests.

Keywords: SPARQL · Link traversal–based query processing · web of linked data

1 Introduction

The World-Wide Web provides a permissionless information space organized as interlinked documents. The Semantic Web builds on top of it by representing data in a machine-interpretable format, fueled by the Linked Data principles. In contrast to more complex data-driven APIs, the simplicity of document-based interfaces comes with multiple advantages. They scale easily, and can be hosted on many different kinds of hardware and software; we can realize the *"anyone can say anything about anything"* principle because every publisher has their own domain in the Web, within which they can freely refer to concepts from other domains; and complex features such as access control or versioning are technically easy to achieve on a per-document basis.

However, decentralized interfaces are notoriously more difficult to query. As such, the past decade has instead been characterized by Big Data and hypercentralization, in which data from multiple sources becomes aggregated in an increasingly smaller number of sources. While extremely powerful from a query and analytics perspective, such aggregation levels lead to a loss of control and freedom for individuals and small- to medium-scale data providers. This in turn has provoked some fundamental legal, societal, and economical questions regarding the desiredness of such hypercentral platforms.

As such, there is again an increasing demand for more decentralized systems, where data is stored closer to its authentic source, in line with the original intentions of the Web [14].

As with Big Data, query processing on the Semantic Web has traditionally focused on *single* databases. The SPARQL query language allows querying such a single RDF store through the SPARQL protocol, which places significantly more constraints on the server than a document-based interface [15]. While *federated* query processing enables incorporating data from multiple SPARQL endpoints, federated queries have very limited link traversal capabilities and SPARQL endpoints easily experience performance degradation [2].

Fortunately, a technique was introduced to query webs of data: *Link Traversal–based Query Processing* (LTQP) [5, 8], in which an agent evaluates a SPARQL query over a set of documents that is continuously expanded by selectively following hyperlinks inside of them. While LTQP demonstrates the independence of queries and selection of sources (on which these queries need to be executed), it has mostly remained a theoretical exercise, as its slow performance makes it unsuitable for practical purposes. The fact that LTQP can yield more results than single-source query evaluation, gave rise to different notions of *query semantics* and *completeness* [9]. While more data can be considered advantageous, it can also lead to doubts regarding *data quality*, *trustworthiness*, or *license compatibility*. Together with performance, these concerns seem to have pushed LTQP to the background.

In this article, we identify two limitations of existing LTQP approaches. Essentially, all existing LTQP approaches identify a *subweb* of the web of linked data on which a query needs to be executed. The first limitation is that *the responsibility for defining how to construct this subweb is entirely in the hands of the data consumer, from now on referred to as the* ***querying agent*** (which can be an end-user or machine client). In other words, existing approaches make the assumption that the querying agent can determine perfectly which links should be traversed. However, since every data publisher can freely choose how to organize their data, we cannot expect a single agent to possess complete knowledge of how such traversals should proceed. A second restriction is that current LTQP formalisms provide an all-or-nothing approach: a document is either included in the subweb of interest in its entirety, or not at all, while for data-quality reasons, it would be useful to only take parts of documents into account. For instance, an academic who has moved institutions might specify that the data provided by institution A is trustworthy up to a certain date and that for later information about them, institution B should be consulted. More radically, a certain end user might wish to specify that Facebook's data about who her friends are is correct, without thereby implying that any triple published by Facebook should be taken into account when performing a query.

In this paper, building on the use case of the next section, we propose an approach for *guided* link traversal that overcomes these two limitations. In our proposal, each data publisher has their own subweb of interest, and publishes a specification of how it can be constructed. They can use this for instance to describe the organization of their data, or to describe parties they trust (as well as for which data they trust them). The data consumer can then construct a subweb of interest *building on* the subwebs of the publishers, e.g., deciding to include parts of a subweb, or to omit it. As such, the data publishers *guide* the data consumer towards relevant data sources. We focus on the theoretical foundations and highlight opportunities for result quality and performance improvements.

```
<https://uma.ex/#me> foaf:knows
  <https://ann.ex/#me>, <https://bob.ex/#me>.
<https://bob.ex/#me> foaf:img <bob.jpg>.
```

Document 1: Contents of *https://uma.ex/*

```
<https://ann.ex/#me> foaf:name "Ann";
  foaf:mbox <mailto:ann@corp.ex>;
  foaf:img <me.jpg>.
```

Document 4: Contents of *https://corp.ex/ann/*

```
<https://ann.ex/#me> foaf:isPrimaryTopicOf <https://corp.ex/ann/>.
<https://ann.ex/#me> foaf:weblog <https://ann.ex/blog/>.
<https://ann.ex/#me> foaf:maker <https://photos.ex/ann/>.
```

Document 2: Contents of *https://ann.ex/*

```
<https://bob.ex/#me> foaf:name "Bob";
  foaf:mbox <mailto:me@bob.ex>;
  foaf:img <funny-fish.jpg>.
<https://uma.ex/#me> foaf:knows
  <http://dbpedia.org/resource/Mickey_Mouse>.
<https://ann.ex/#me> foaf:name "Felix".
```

Document 3: Contents of *https://bob.ex/*

```
SELECT ?friend ?name ?email ?picture WHERE {
  <https://uma.ex/#me> foaf:knows ?friend.
  ?friend foaf:name ?name.
  OPTIONAL { ?friend foaf:mbox ?email.
             ?friend foaf:img  ?picture. }
}
```

Query 1: Application query in SPARQL

	?friend	?name	?email	?picture
1	<https://ann.ex/#me>	"Ann"	<mailto:ann@corp.ex>	<https://corp.ex/ann/me.jpg>
2	<https://bob.ex/#me>	"Bob"	<mailto:me@bob.ex>	<https://uma.ex/bob.jpg>
3	<https://bob.ex/#me>	"Bob"	<mailto:me@bob.ex>	<https://bob.ex/funny-fish.jpg>
4	<https://ann.ex/#me>	"Felix"	<mailto:ann@corp.ex>	<https://corp.ex/ann/me.jpg>
5	dbr:Mickey_Mouse	"Mickey Mouse"@en	*NULL*	*NULL*

Results 1: Possible results of LTQP of the query in Query 1 with *https://uma.ex/* as seed

2 Use Case

As a guiding example throughout this article, we introduce example data and queries for a use case that stems from the Solid ecosystem [14], where every person has their own *personal data vault*. Let us consider 3 people's profile documents, stored in their respective data vaults. Uma's profile (Document 1) lists her two friends Ann and Bob. Ann's profile (Document 2) contains links to her corporate page and various other pages. Bob, a self-professed jokester, lists his real name and email address in his profile (Document 3), in addition to a funny profile picture and a couple of factually incorrect statements (which he is able to publish given the open nature of the Web). Note how Ann provides additional facts about herself into the external document she links to (Document 4), and Uma's profile suggests a better profile picture for Bob (Document 1).

Next, we consider an *address book* application that displays the details of a user's contacts. At design-time, this application is unaware of the context and data distribution of the user and their friends. If we assume Uma to be the user, then the application's data need can be expressed as Query 1, which is a generic SPARQL template in which only the URL corresponding to Uma's identity (*https://uma.ex/#me*) has been filled out.

With traditional LTQP (under c_{All} semantics [9]), results include those in Results 1. However, the actually desired results are Rows 1 and 2, which contain Uma's two friends with relevant details. Rows 3–5 are formed using triples that occur in Bob's profile document but are not considered trustworthy by Uma (even though other triples in the same document are). To obtain these results, a query engine would need to fetch at least 7 documents: the profile documents of the 3 people (Uma, Ann, Bob), the 3 documents referred to by Ann's profile (Document 2), and the DBpedia page for Mickey Mouse.

3 Preliminaries

As a basis for our data model of a Web of Linked Data, we use the RDF data model [3]. That is, we assume three pairwise disjoint, infinite sets: $\mathcal{U}$ (for URIs), $\mathcal{B}$ (for blank nodes), $\mathcal{L}$ (for literals). An *RDF triple* is a tuple $(s, p, o) \in \mathcal{T}$, with $\mathcal{T}$ the set of all triples defined as $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$; if $t = (s, p, o) \in \mathcal{T}$, then $\text{uris}(t) = \{s, p, o\} \cap \mathcal{U}$. A set of triples is called a *triple graph* or *RDF graph*. An *RDF dataset* is a set of tuples $\{\langle n_i, g_i \rangle\}$ where $n_i \in \mathcal{U}$ and g_i an RDF graph, where g_0 is referred to as the *default graph*.

We assume another set $\mathcal{D}$, disjoint from the aforementioned sets $\mathcal{U}$, $\mathcal{B}$ and $\mathcal{L}$, whose elements are referred to as *documents*. The RDF graph contained in each document is modeled by a function $data : \mathcal{D} \to 2^{\mathcal{T}}$ that maps each document to a finite set of triples.

Definition 1. *A* Web of Linked Data (WOLD) *W is a tuple $\langle D, data, adoc \rangle$ where D is a set of documents $D \subseteq \mathcal{D}$, $data$ a function from D to $2^{\mathcal{T}}$ such that $data(d)$ is finite for each $d \in D$, and $adoc$ a partial function from $\mathcal{U}$ to D. If W is a WOLD, we use D_W, $data_W$, and $adoc_W$ for its respective components. The set of all WOLDs is denoted $\mathcal{W}$.*

We aim to define parts of a web as subwebs. While existing definitions only consider the inclusion of documents in their entirety [9], we allow for *partial* documents to enable fine-grained control about which data is to be used for answering certain queries.

Definition 2. *Consider two WOLDs $W = \langle D, data, adoc \rangle$ and $W' = \langle D', data', adoc' \rangle$. We say that W' is a* subweb *of W if i) $D' \subseteq D$, ii) $\forall d \in D' : data'(d) \subseteq data(d)$, and iii) $adoc'(u) = adoc(u)$ if $adoc(u) \in D'$ and $adoc'(u)$ is undefined otherwise.*
We write $subwebs(W)$ for the set of subwebs of W.

The simplest type of subwebs are those only consisting of a single document.

Definition 3. *Let W be a WOLD and $d \in D$. We use $singleton(d, W)$ to denote the (unique) subweb $\langle \{d\}, data', adoc' \rangle$ of W with $data'(d) = data(d)$.*

Additionally, if two subwebs of a given WOLD are given, we can naturally define operators such as union and intersection on them; in this paper, we will only need the union.

Definition 4. *If W_1 and W_2 are subwebs of W, we define $W_1 \cup W_2$ to be the unique subweb $\langle D', data', adoc' \rangle$ of W with*

- *$D' = D_{W_1} \cup D_{W_2}$, and*
- *$data'(d) = data_{W_1}(d) \cup data_{W_2}(d)$ for each $d \in D'$, where, slightly abusing notation, we use $data_{W_i}(d) = \emptyset$ if $d \notin D_{W_i}$.*

4 Requirements

From the use case, we extracted four requirements that motivate our definitions.

A Declarative Language for Selecting Data Sources Similar to existing LTQP approaches, we need a language to describe which data sources to select (possibly starting from a given seed). We want such a language to be declarative, i.e., focus on *which* sources to use, rather than *how* to obtain them. Formally, we expect a source selection expression to evaluate in a given WOLD to a set of URIs representing the documents to be included.

Independence of Query and Subweb Specification Motivated by principles of reusability and separation of concerns, we want the *query* to be formulated independently from the *subweb over which the query is to be evaluated*. While it might — to a certain extent — be possible to encode traversal directions in (federated) SPARQL queries, *what do I want to know* and *where do I want to get my information* are two orthogonal concerns that we believe should be clearly separated, in order to improve readability, maintainability, and reusability. E.g., in the use case, the phone book application defines the *query*, while Uma defines her own *subweb of interest* (consisting of her own document, as well as parts of the documents of her friends). The application should be able to run with different subwebs (e.g., coming from other users), and Uma's subweb of interest should be reusable in other applications.

Scope Restriction of Sources One phenomenon that showed up in the use case is that we want to trust a certain source, but only for specific data. We might for instance want to use all our friends' data sources, but only to provide information about themselves. This would avoid "faulty" data providers such as Bob to publish data that pollute up the entire application, and it would give a finer level of control over which data is to be used to answer queries. On the formal level, this requirement already manifests itself in the definition of *subweb* we chose: contrary to existing definitions [9], we allowed a document in a subweb to have only a subset of the data of the original document.

Distributed Subweb Specifications Finally, we arrive at the notion of distribution. This is the feature in which our approach most strongly deviates from the state-of-the-art in link traversal. While the semantic web heavily builds on the assumption that *data* is decentralized and different agents have different pieces of data to contribute, existing link traversal–based approaches still assume that the *knowledge of where this data can be found* is completely in the hands of the querying agent at query time, or at least that the *principles by which the web has to be traversed* can be described by the querying agent. However, as our use case illustrates, this is not always the case: Ann decided to distribute her information over different pages; the agent developing the phone book application cannot possibly know that the triple `<https://ann.ex/#me> foaf:isPrimaryTopicOf <https://corp.ex/ann/>.` indicates that information from `<https://corp.ex/ann/>` is "equally good" as information from Ann's main document. Stated differently, only Ann knows how her own information is organized and hence if we want to get personal information from Ann, we would want her to be able to describe herself how or where to find this data. To summarize, we aim to allow *document publishers to publish specifications of subwebs in the same declarative language as used by query agents* and *query agents to decide whether or not to include the data from such subwebs*.

5 Related Work

Link Traversal-based Query Processing Over a decade ago, the paradigm of Link Traversal-based Query Processing was introduced [8], enabling queries over document-oriented interfaces. The main advantage of this approach is that queries can always be executed over live data, as opposed to querying over indexed data that may be

stale. The main disadvantages of this approach are that query termination and result completeness are not guaranteed, and that query execution is typically significantly slower than database-centric approaches such as SPARQL endpoints. Several improvements have been suggested to cope with these problems [5]. For example, the processing order of documents can be changed so that certain documents are *prioritized* [10], which allows relevant results to be emitted earlier in an iterative manner [6], but does not reduce total execution time. In this work, we propose to tackle this problem by allowing publishers to specify their subweb of interest. These specifications are then used to *guide* the query engine towards relevant (according to the data publishers at hand) documents.

Reachability Semantics The SPARQL query language was originally introduced for query processing over RDF databases. Since LTQP involves a substantially different kind of sources, a family of new semantics was introduced [9], involving the concept of a *reachable* subweb. When executing a query over a set of *seed documents*, the reachable Web is the set of documents that can be reached from these seeds using one of different *reachability criteria*. These criteria are functions that test each data triple within retrieved documents, indicating which (if any) of the URIs in the triple components should be dereferenced by interpreting them as the URI of a document that is subsequently retrieved over HTTP. The c_{All} reachability criterion involves following all encountered URIs, which is the strategy in the example of Results 1. A more elaborate criterion is c_{Match}, which involves following URIs from data triples that match at least one triple pattern from the query. c_{Match} can significantly reduce the number of traversals compared to c_{All}. However, evaluating Query 1 with c_{Match} semantics would not yield results for Ann (rows 1 and 4). Her details are only reachable via a triple with predicate `foaf:isPrimaryTopicOf`, which does not match any of the query's triple patterns; hence, the relevant document is never visited. So while c_{Match} can lead to better performance, it comes at the cost of fewer results, showing that none of these approaches are optimal.

Delegation The concept of subwebs is somewhat related to the presence of active rules in rule-based languages for distributed data management. A particularly relevant project in this context is Webdamlog [1], a Datalog-based declarative language for managing knowledge on the web with support for rule-delegation. Here, delegation is achieved by allowing rules to get partially materialized by different peers.

6 A Formalism for Subweb Specifications

Inspired by the desired properties from Section 4, we now define a formalism to describe subwebs of interest. In our formalism, different agents will be able to provide a description of a *subweb of interest*; they will be able to specify declaratively in (which parts of) which documents they are interested. We do not make any assumption here about what the reason for this "interest" is; depending on the context at hand, different criteria such as relevance, trustworthiness, or license-compatibility can be used. Such a description of a subweb of interest can be given by the **querying agent** (an end-user or machine client) which provides it at runtime to the **query processor**. Additionally, every **data publisher** can use the same mechanism to make assertions about her beliefs, such that other

data publishers or querying agents can reuse those instead of requiring explicit knowledge. For instance, a data publisher can express which sources they consider relevant or trustworthy for what kinds of data: a researcher might indicate that a certain source represents her publication record correctly, whereas another source captures her affiliation history. A certain agent *might* or *might not* choose to take the subweb of interest of a data publisher into consideration. In the use case of Section 2, the application generates a query P as Query 1, and end-user Uma expresses she trusts her own profile for her list of contacts, and to trust those contacts for their own details. Furthermore, each of these friends can indicate which other documents they trust for which information. For instance, Ann could express that she trusts *corp.ex* for her personal details. Essentially, in this case Uma partially *delegates* responsibility of traversing the web to Ann, but only retains information about Ann from Ann's subweb of interest. This leads to the following definitions.

Definition 5. *A* source selector *is a function* $\sigma : \mathcal{W} \to 2^{\mathcal{U}}$. *A* filter *is a function* $f : 2^{\mathcal{T}} \times \mathcal{U} \to 2^{\mathcal{T}}$ *such that* $f(S,u) \subseteq S$ *for every* $S \subseteq \mathcal{T}$ *and* $u \in \mathcal{U}$. *For a* WOLD $W = \langle D, data, adoc\rangle$ *and* URI u*; we extend the notation and also write* $f(W,u)$ *to denote the subweb* $\langle D, data', adoc\rangle$ *of* W *with* $data'(d) := f(data(d),u)$ *for each* $d \in D$.

In our running example, if Uma wants for each of her friends to only include statements they make about themselves, she can use a source selector σ that extracts her friends, e.g, with $\sigma(W) = \{o \mid \langle s, \texttt{foaf:knows}, o\rangle \in data(adoc(s))$ with $s = \texttt{<https://uma.ex/\#me>}\}$ and with a filter that maps (S,u) to $\{\langle s,p,o\rangle \in S \mid s = u\}$. If we assume that W is a WOLD in which only a particular friend u of Uma provides triples, then $f(W,u)$ is the subweb of W in which friend u has only the triples making statements about him or herself.

Definition 6. *A* subweb specification*, often denoted* Θ*, is a set of tuples of the form* (σ, b, f)*, where* σ *is a source selector;* b *is a Boolean; and* f *is a filter.*

Intuitively, the Boolean b in (σ, b, f) indicates whether to include for each URI $u \in \sigma(W)$ (the filtered version of) the subweb of $adoc(u)$ or only u's document. Finally, this brings us to the definition of a specification-annotated WOLD (sa-WOLD in short): a WOLD extended with the construction rules of all data publishers.

Definition 7. *A* specification-annotated WOLD *is a tuple* $\mathbb{W} = \langle W, \boldsymbol{\Theta}\rangle$ *consisting of a* WOLD $W = \langle D, data, adoc\rangle$ *and an associated family* $\boldsymbol{\Theta} = (\Theta_d)_{d\in D}$ *of subweb specifications.*

In a sa-WOLD, each data publisher declares her subweb specification that can be used to construct her subweb of interest. The value of a subweb specification in a sa-WOLD is defined as follows:

Definition 8. *Let* $\mathbb{W} = \langle W, \boldsymbol{\Theta}\rangle$ *be a sa-*WOLD *with* $W = \langle D, data, adoc\rangle$*, and* Θ *a subweb specification. Then,* $[\![\Theta]\!]^{\mathbb{W}}$ *denotes the subweb specified by* Θ *for* $\mathbb{W}$*,*

$$[\![\Theta]\!]^{\mathbb{W}} := \bigcup_{(\sigma,b,f)\in\Theta \wedge u\in\sigma(W)} f\left(singleton(adoc(u),W) \cup \left([\![(\Theta_{adoc(u)})]\!]^{\mathbb{W}} \textbf{ \textit{if} } b\right), u\right),$$

where $(S \textbf{ \textit{if} } b)$ *equals* S *if* b *is true and the empty* WOLD *(the unique* WOLD *without documents) otherwise. The* subweb of interest of a document $d \in D$ in $\mathbb{W}$ *is defined as* $soi(d,\mathbb{W}) := singleton(d,W) \cup [\![\Theta_d]\!]^{\mathbb{W}}$.

Since not just the data publishers, but also the querying agents should be able to specify a subweb of interest, we naturally obtain the following definition.

Definition 9. *A* specification-annotated query *is a tuple* $\mathbb{P} = \langle P, \Theta \rangle$ *with* P *a* SPARQL *query and* Θ *a subweb specification. The* evaluation *of* $\mathbb{P}$ *in* $\mathbb{W}$*, denoted* $[[\mathbb{P}]]^{\mathbb{W}}$*, is defined by* $[[\mathbb{P}]]^{\mathbb{W}} := [[P]]^{[[\Theta]]^{\mathbb{W}}}$

Here, we use $[[P]]^{W'}$ to denote the evaluation of the SPARQL query in the dataset that is the union of all the documents in W' (to be precise, this is the RDF dataset with as default graph the union of all the data in all documents of the subweb, and for each URI u with $adoc(u) = d$ a named graph with name u and as triples the data of d). Of course, we need a mechanism to *find* all those documents, which is what Θ will provide.

In the next section, we propose a concrete SPARQL-based instantiation of the theoretical framework presented here and illustrate our use case in that setting. Afterwards, we will formally compare our proposal to existing LTQP approaches.

7 Expressing Subweb Specifications

In this section, we propose a syntax for subweb specifications (as formalized in Section 6), named the Subweb Specification Language (SWSL), inspired by LDQL and SPARQL. In order to lower the entry barrier of this syntax to existing SPARQL engine implementations, we deliberately base this syntax upon the SPARQL grammar. This enables implementations to reuse (parts of) existing SPARQL query parsers and evaluators.

The grammar below represents the SWSL syntax in Extended Backus–Naur form (EBNF) with start symbol ⟨start⟩. The specifications begin with the **`FOLLOW`** keyword, followed by a ⟨sources⟩ clause, an *optional* **`WITH SUBWEBS`** keyword, and an *optional* ⟨filter⟩ clause.

⟨start⟩ ::= **`FOLLOW`** ⟨sources⟩ [**`WITH SUBWEBS`**] [⟨filter⟩]
⟨sources⟩ ::= ⟨variables⟩ **`{`** ⟨GroupGraphPattern⟩ **`}`** [⟨recurse⟩]
⟨variables⟩ ::= **`?`**⟨VARNAME⟩ | **`?`**⟨VARNAME⟩ ⟨variables⟩
⟨recurse⟩ ::= **`RECURSE`** [⟨INTEGER⟩]
⟨filter⟩ ::= **`INCLUDE`** ⟨ConstructTemplate⟩ [**`WHERE {`** ⟨GroupGraphPattern⟩ **`}`**]

Intuitively, a full SWSL expression corresponds to a single subweb specification tuple (σ, b, f) where the ⟨sources⟩ clause correspond to the source selection function σ, the keyword **`WITH SUBWEBS`** corresponds to the Boolean b, and the ⟨filter⟩ clause corresponds to the filter function f. We explain each of these parts in more detail hereafter.

Selection of Sources The ⟨sources⟩ will be evaluated in the context of a set S of seed documents. For subweb specifications provided to the query processor, this set of seeds will be given explicitly, whereas for subweb specifications found in a document, the set S is the URI of that document . A ⟨sources⟩ clause begins with a list of SPARQL variables, followed by a source extraction expression defined as SPARQL's ⟨GroupGraphPattern⟩ clause. The output is a set of bindings of the given variables, indicating URIs whose documents are to

be included. For instance, when evaluating the expression $\texttt{?}v_1 \ldots \texttt{?}v_n$ **{** G **}** in a WOLD W with seed set S, the resulting source selection is

$$\sigma(W) = \bigcup_{u \in S} \{\mu(v_i) \in \mathcal{U} \mid 1 \leq i \leq n \wedge \mu \in [\![G]\!]^{data(adoc(u))}\},$$

where $[\![G]\!]^{DS}$ is the evaluation of the GroupGraphPattern G on a dataset DS.

Recurring Source Selection A ⟨sources⟩ clause may have at the end an optional ⟨recurse⟩ clause. If **RECURSE** is not used in a specification, then this latter will only apply to the document in which it is defined; else, the specification will apply to that document, and all output URIs, taken as seed (recursively). In other words, the ⟨sources⟩ clause will be applied to all documents that are obtained when following a chain of one or more links using the specification. The ⟨recurse⟩ clause has an optional nonnegative integer parameter, which indicates the maximum recursion *depth*. A depth of 0 is equivalent to not defining the ⟨recurse⟩ clause. A depth of m means that all documents that are obtained when following a link path of length m from the seeds are considered. This recursion capability calls for the need to express *the current document's URI*. To achieve this, SWSL syntax reuses SPARQL's relative IRI capability. Concretely, every time an SWSL specification is applied on a document, the document's URI will be set as base IRI to the SWSL specification, so that relative IRIs can be resolved upon this IRI.

Inclusion of Subwebs of Selected Sources This is determined by the optional keyword **WITH SUBWEBS**. Thus, if an SWSL specification has the **WITH SUBWEBS** option, this is equivalent to a subweb specification tuple with b is true. Otherwise, b is false.

Document Filtering The ⟨filter⟩ clause is an optional clause indicating that only certain parts of the document are considered. Without this clause, the entire document is included. The ⟨filter⟩ clause is similar to SPARQL's ⟨ContructQuery⟩ clause. It exists in *compact* or *extended* forms; in the latter, filtering constraints can be added via **WHERE** keyword.

Concretely, the extended form is defined by the SPARQL's ⟨ConstructTemplate⟩ and ⟨GroupGraphPattern⟩ productions. The ⟨ConstructTemplate⟩ acts as a template of triples to accept, while the ⟨GroupGraphPattern⟩ imposes conditions to do so. It is also possible that in the bodies of the ⟨GroupGraphPattern⟩ and ⟨ConstructTemplate⟩ there are variables that are mentioned in the ⟨GroupGraphPattern⟩ of ⟨sources⟩ clause. This implies that they should be instantiated according to the result of the first ⟨GroupGraphPattern⟩.

The compact form is defined by ⟨ConstructTemplate⟩, which acts as syntactical sugar to the extended with an empty ⟨GroupGraphPattern⟩. Thus, to define ⟨filter⟩ clause's semantics, we only need the extended form. To illustrate this, consider an expression

FOLLOW $\texttt{?}v_1$ **{** G_1 **}** **INCLUDE** C **WHERE** **{** G_2 **}**

We already saw that when evaluated in context u, this induces a source selector selecting those v such that $\mu_1(\texttt{?}v_1) = v$, for some $\mu_1 \in [\![G_1]\!]^{data(adoc(u))}$. The associated filter is

$$f(S, v) = \bigcup_{\mu_1 \in [\![G_1]\!]^{data(adoc(u))} \mid \mu_1(\texttt{?}v_1) = v} \{t \in S \mid t \in [\![\mu_2(\mu_1(C))]\!]^S \text{ for some } \mu_2 \in [\![\mu_1(G_2)]\!]^S\}$$

```
<https://uma.ex/#me> ex:hasSpecification <#spec1>.
<#spec1> ex:appliesTo <https://uma.ex/>;
         ex:scope """
          FOLLOW ?friend WITH SUBWEBS {
            <https://uma.ex/#me> foaf:knows ?friend.
          } INCLUDE { ?friend ?p ?o. }
         """^^ex:SWSL.
```

Listing 1: Subweb Specification of https://uma.ex/

```
<https://ann.ex/#me> ex:hasSpecification <#spec2>.
<#spec2> ex:appliesTo <https://ann.ex/>;
              ex:scope """
               FOLLOW ?page {
                 ?topic foaf:isPrimaryTopicOf ?page.
               } INCLUDE { ?topic ?p ?o. }
              """^^ex:SWSL.
```

Listing 2: Subweb Specification of https://ann.ex/

Expressing Document Subwebs In this work, we assume that each published document can link to its own context where they indicate the documents they consider relevant using an SWSL subweb specification. For illustration, we consider the predicate *ex:hasSpecification* that is attached to the current document. An *ex:Specification* is a resource that contains at least a value for *ex:scope*, pointing to one or more SWSL strings. This resource can also contain metadata about the subweb specification.

Application to the Use Case Listing 1 shows a part of Uma's profile where she exposes an SWSL subweb specification to indicate that her friends can express information about themselves. This specification states that all *foaf:knows* links from Uma should be followed, and that from those followed documents, only information about that friend should be included. By **`WITH SUBWEBS`**, she indicates that her friends' subwebs must be included in her subweb. Then, Ann can express in her subweb specification (Listing 2) that she trusts documents pointed to by *foaf:isPrimaryTopicOf* links about triples about the topic she indicates. With these subweb specifications, Query 1 produces only Rows 1–3 of Results 1. However, we still include the non-desired profile picture from Bob in our results (Row 3). Extending the notion of filter to also allow this is left for future work.

8 Power and Limitations of Existing LTQP Approaches

Since LDQL is a powerful link traversal formalism that has been shown to subsume other approaches such as reachability-based querying [4], this raises the question: to which extent can LDQL in itself achieve the requirements set out in Section 4? In the current section we formally investigate this, after introducing some preliminaries on LDQL.

8.1 Preliminaries: LDQL

LDQL is a querying language for linked data. Its most powerful aspect is the navigational language it uses for identifying a subweb of the given WOLD. The most basic block that constitutes LDQL's navigational language is a *link pattern* that is a tuple in $(\mathcal{U} \cup \{\_, +\}) \times (\mathcal{U} \cup \{\_, +\}) \times (\mathcal{U} \cup \mathcal{L} \cup \{\_, +\})$. Intuitively, a link pattern requires a context uri u_{ctx}, then evaluates to a set of URIs (the links to follow) by matching the link pattern against the triples in the document that u_{ctx} is authoritative for. Formally, we say that a link pattern $lp = \langle \ell_1, \ell_2, \ell_3 \rangle$ *matches* a triple $\langle x_1, x_2, x_3 \rangle$ with result u in the context of a URI u_{ctx} if the following two points hold: i) there exists $i \in \{1, 2, 3\}$ such that $\ell_i = \_$ and $x_i = u$, and ii) for every $i \in \{1, 2, 3\}$ either $\ell_i = x_i$, or $\ell_i = +$ and $x_i = u_{ctx}$, or $\ell_i = \_$

Link patterns are used to build *link path expressions* (LPEs) with the following syntax:

$$lpe := \varepsilon \mid lp \mid lpe/lpe \mid lpe \,|\, lpe \mid lpe^* \mid [lpe]$$

Table 1: Value of link path expressions

lpe	$[\![lpe]\!]^u_W$
ϵ	$\{u\}$
lp	$\{u' \mid lp$ matches with $t)$ with result u' in context u for some $t \in data(adoc(u))\}$
lpe_1/lpe_2	$\{v \mid v \in [\![lpe_2]\!]^{u'}_W$ and $u' \in [\![lpe_1]\!]^u_W\}$
$lpe_1 \mid lpe_2$	$[\![lpe_1]\!]^u_W \cup [\![lpe_2]\!]^u_W$
lpe^*	$\{u\} \cup [\![lpe]\!]^u_W \cup [\![lpe/lpe]\!]^u_W \cup [\![lpe/lpe/lpe]\!]^u_W \cup ...$
$[lpe]$	$\{u \mid [\![lpe]\!]^u_W \neq \emptyset\}$

where lp is a link pattern. In a given WOLD W, the value of a link path expression lpe in context URI u (denoted $[\![lpe]\!]^u_W$) is a set of URIs as given in Table 1.

An LDQL query is a tuple $q = \langle lpe, P\rangle$ with lpe a link path expression and P a SPARQL query. The value of such a query q in a WOLD W with a set of seed URIs S is

$$[\![q]\!]^S_W := [\![P]\!]^{W'} \text{ where } W' = \bigcup_{s\in S, u\in[\![lpe]\!]^s_W} singleton(adoc(u), W),$$

i.e., the query P is evaluated over the (RDF dataset constructed from the) data sources obtained by evaluating the link path expression starting in one of the seeds.

Remark 1. [11] allows one other form of link path expression, where an entire LDQL query is nested in in an LPE; for the purpose of this paper, we opt to use a strict separation between query and source selection and omit this last option[4]. Additionally, they consider (Boolean) combinations of queries, thereby allowing to use different LPEs for different parts of the expression; we briefly come back to this when discussing scope restriction.

8.2 LDQL and the Requirements

A Declarative Language for Selecting Data Sources In LDQL, the link path expressions provide a rich and flexible declarative language for describing source selection. Here, paths through the linked web are described using a syntax similar to regular expressions. For instance, the LDQL expression $\langle$+, `foaf:knows`, _$\rangle/\langle$+, `foaf:knows`, _$\rangle$ when evaluated in a given URI u (the context) traverses to u's friends f (as explicated by triples of the form `<u,foaf:knows,f>` in $adoc(u)$) and subsequently to their friends f_2 (as indicated by triples `<f,foaf:knows,f2>` in $adoc(f)$). In other words, this example expression identifies the documents of friends of friends of a given person.

Independence of Query and Subweb Specification The design philosophy behind LDQL does not start from an independence principle similar to the one proposed here. That is, in its most general form, LDQL allows intertwining the source selection and the query. For instance, the LDQL query $\langle lpe_1, P_1\rangle$ AND $\langle lpe_2, P_2\rangle$ expresses the SPARQL query P_1 AND P_2, and on top of that specifies that different parts of the query should be evaluated with respect to different sources, and hence violating our principle of independence. However, independence can easily be achieved in LDQL by only considering LDQL queries of the form $\langle lpe, P\rangle$ with lpe a link path expression and P a SPARQL query.

[4] Notably, this option was also not present in the original work [7].

Scope Restriction of Sources The semantics of an LDQL query $\langle lpe, P\rangle$ is obtained by first evaluating lpe starting from a seed document s, resulting in a set of URIs $[\![lpe]\!]^s_W$; the SPARQL query P is then evaluated over the union of the associated documents. That is, to compute the result of $\langle lpe, P\rangle$, for each document $adoc(u)$ with $u \in [\![lpe]\!]^s_W$, its entire content is used. As such, LDQL provides no mechanism for partial inclusion of documents. However, while LDQL cannot select *parts of documents*, it *can* be used, as discussed above, to apply source selection strategies only to *parts of queries* and thereby to a certain extent achieve the desired behaviour. E.g., the query $\langle lpe_1, (?x, \texttt{foaf:knows}, ?y)\rangle$ AND $\langle lpe_2, (?y, \texttt{foaf:mbox}, ?m\rangle$ will only use triples with predicate `foaf:knows` from documents produced by lpe_1. However, this sacrifices the independence property, and for complex queries and filters, this is not easy to achieve.

Distributed Subweb Specifications This now brings us to the main topic of this section: studying to which extent it is possible in LDQL to distribute the knowledge of how to construct the subweb of interest and as such to *guide* the data consumer towards interesting/relevant documents. To answer this question, we will consider a slightly simplified setting, without filters (all filters equal the identity function *id* on their first argument) and where the Boolean b in (σ, b, f) is always true. I.e., each agent states that they wish to include the complete subweb of interest of all URIs identified by σ. In this setting, we wonder if data publishers can, instead of publishing their subweb specification *in addition to* their regular data, encode their subweb specification as triples *in* the document (as meta-information), and use *a single* "meta" link path expression that interprets these triples for the traversal. This is formalized as follows.

Definition 10. *Let $\mathcal{S}$ be a set of source selectors, $enc : \mathcal{S} \to 2^{\mathcal{T}}$ a function mapping source selectors σ onto a set of triples $enc(\sigma)$, and $\mathbb{W} = \langle\langle D, data, adoc\rangle, \mathbf{\Theta}\rangle$ a sa-WOLD in which each subweb specification is of the form $(\sigma, \text{true}, id)$ with $\sigma \in \mathcal{S}$. The* encoding of $\mathbb{W}$ *by enc is the WOLD $enc(\mathbb{W}) = \langle D, data', adoc\rangle$ with for each $d \in D$:*

$$data'(d) = data(d) \cup \bigcup_{\{\sigma \mid (\sigma, \text{true}, id) \in \Theta_d\}} enc(\sigma).$$

Definition 11. *Let $\mathcal{S}$ be a set of source selectors, enc a function $\mathcal{S} \to 2^{\mathcal{T}}$, and e_{meta} an LPE. We say that (enc, e_{meta})* captures *$\mathcal{S}$ if for each sa-WOLD in which subweb specifications only use triples of the form $(\sigma, \text{true}, id)$ with $\sigma \in \mathcal{S}$ and for each URI u,*

$$[\![e_{meta}]\!]^u_{enc(\mathbb{W})} = soi(adoc(u), \mathbb{W}).$$

We will say that LDQL can capture distribution of functions in *$\mathcal{S}$ if there exist some enc and e_{meta} that capture $\mathcal{S}$.*

To define the encodings, we will make use of some "fresh" URIs we assume not to occur in any WOLD. In our theorems, we will make use of some specific sets of source selectors. A source selector σ is *constant* if it maps all WOLDs onto the same set of URIs, i.e., if $\sigma(W) = \sigma(W')$ for all WOLDs W, W'; the set of all constant source selectors is defined as $\mathcal{S}_{const}$. If p and u are URIs, we define the source selector $all_{p^*,u}$ as follows:

$$all_{p^*,u} : W \mapsto [\![(+, p, \_)^*]\!]^u_W.$$

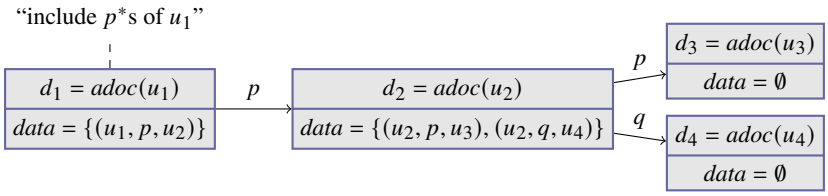

Fig. 1: Example WOLD used in LDQL inexpressivity proof.

Intuitively, the function $all_{p^*,u}$ identifies the set of all ps of ps of .... of u. For instance, by taking $p = friend$, we include all direct or indirect friends of u. For a fixed p, we write $\mathcal{S}_{p^*}$ for the set of source selectors $all_{p^*,u}$. We write $\mathcal{S}_*$ for the set of all source selectors of the form $all_{p^*,u}$ for any p. The set $\mathcal{S}_*$ allows each data publisher to choose her own strategy for constructing the subweb, e.g., one data publisher might include all her $friend^*$s, another her $colleague^*$s and a third one only URIs explicitly trusted (i.e., their $trust^*$s). Our main expressivity results are then summarized as follows:

Theorem 1. *LDQL captures distribution of $\mathcal{S}_{const}$ and $\mathcal{S}_{p^*}$, but not of $\mathcal{S}_*$.*

Proof (Sketch of the proof). For the positive results, we can provide an explicit encoding and meta-expression. For instance for showing that it captures $\mathcal{S}_{p^*}$ for a given p, we can take $enc(all_{p^*,u}) = \{(a, a, u)\}$ and $e_{meta} = ((a, a, \_)/(+, p, \_)^*)^*$ with a a fresh URI. In this expression e_{meta}, the link pattern $(a, a, \_)$ is used to navigate the u whose ps of ps of... we wish to include; the part $(+, p, \_)^*$ then navigates to all such p^*s. The outermost star ensures that for each u that is found, also their subweb of interest is included.

The proof of the negative result relies heavily on the fact that an LPE not mentioning p nor q, cannot distinguish the triples (x, p, z) and (x, q, z). If (e_{meta}, enc) were to capture $\mathcal{S}_*$, we can construct a WOLD (see Fig. 1) using only URIs not occurring in e_{meta} in which only one document has a non-empty subweb specification. We then use the aforementioned fact to conclude that $d_3 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$ if and only if $d_4 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$.

9 Discussion

So far, we have studied LTQP from the perspective of data quality; namely, we allow querying agents and/or data publishers to capture a subweb of data that satisfies certain quality properties for them. In real-world applications, such quality properties could for example indicate different notions of trust, or something use-case-specific such as data sensitivity levels. While our formal framework only associates a single subweb specification to each agent, it is not hard to extend it to associate multiple subweb constructions with each agent and allow the querying agent to pick a suitable one.

The same mechanism can be used to improve *efficiency* in two ways: the data publishers can opt to *not* include certain documents in their subweb, and for the ones included, they can use a *filter* which indicates which data will be used from said document.

Most prominently, every publisher of Linked Data typically has their own way of organizing data across documents, and they could capture this structure in their subweb of interest. For example, in contrast to Bob (Document 3), Ann stores her profile

information in multiple documents (Documents 2 and 4). If she were to declare this as a subweb specfication, she can use filters to indicate which data can be found in which documents. A query processor can then exploit this information to only follow links to relevant documents (documents of Ann's subweb for which the filter *could* keep triples that contribute to the query result). For example, Uma's querying agent can use Ann's subweb construction of Listing 2 to prune the set of links to follow, and as such perform a guided navigation while maintaining completeness guarantees. Without even inspecting *https://photos.ex/ann/*, it knows Ann (and thus Uma) does not trust triples in this document for data about her, so fetching it will not change the final query result. Whereas LTQP under c_{All} semantics would require at least 7 HTTP requests, the filters allow us to derive which 4 requests are needed to return all 3 trusted results of the specification-annotated query. Analogous performance gains were observed in work on provenance-enabled queries [16]. In contrast, traditional LTQP cannot make any assumptions of what to encounter behind a link. The work on describing document structures using shapes [12] can be leveraged here.

As such, filters in subweb specifications serve two purposes: they define *semantics* by selecting only part of a data source, and give query processors *guidance* for saving bandwidth and thus processing time.

10 Conclusion

LTQP is generally not considered suitable for real-world applications because of its performance and data quality implications. However, if the current decentralization trend continues, we need to prepare for a future with multi-source query processing, since some data *cannot* be centralized for legal or other reasons.

Federated querying over expressive interfaces such as SPARQL endpoints only addresses part of the problem: empirical evidence suggests that, counterintuitively, less expressive interfaces can lead to faster processing times for several queries [15], while being less expensive to host. A document-based interface is about the simplest interface imaginable, and is thereby partly responsible for the Web's scalability. Hence the need to investigate how far we can push LTQP for internal and external integration of private and public data.

Our formalization for specification-annotated queries creates the theoretical foundations for a next generation of traversal-based (and perhaps *hybrid*) query processing, in which data quality can be controlled tightly, and network requests can be reduced significantly. Moreover, the efforts to realize these necessary improvements are distributed across the network, because every data publisher can describe their own subwebs. Importantly, the availability of such descriptions is also driven by other needs. For instance, initiatives such as Solid [14] store people's personal data as Linked Data, requiring every personal data space to describe their document organization such that applications can read and write data at the correct locations [12].

This article opens multiple avenues for future work. A crucial direction is the algorithmic handling of the theoretical framework, and its software implementation, for which we have ongoing work in the Comunica query engine [13]; an important open question here is how the expressed filters can be exploited for query optimization. Also on the implementation level, the creation and management of subweb specifications should

be facilitated. Empirical evaluations will shed light on cases where subweb annotated WOLDs and queries result in a viable strategy.

Acknowledgements This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. Ruben Taelman and Ruben Verborgh are postdoctoral fellows of the Research Foundation – Flanders (FWO) (1274521N). Heba Aamer is supported by the Special Research Fund (BOF) (BOF19OWB16).

References

1. Abiteboul, S., Bienvenu, M., Galland, A., Antoine, É.: A rule-based language for web data management. In: Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011. pp. 293–304. ACM (2011)
2. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQL Web-querying infrastructure: Ready for action? In: Proceedings of the 12th International Semantic Web Conference. Lecture Notes in Computer Science, vol. 8219, pp. 277–293. Springer (Oct 2013)
3. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and abstract syntax. Recommendation, W3C (Feb 2014), *https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/*
4. Hartig, O.: SPARQL for a Web of Linked Data: semantics and computability. In: Proceedings of the 9th international conference on The Semantic Web: research and applications (May 2012)
5. Hartig, O.: An overview on execution strategies for Linked Data queries. Datenbank-Spektrum **13**(2), 89–99 (2013)
6. Hartig, O.: SQUIN: a traversal based query execution system for the Web of Linked Data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2013)
7. Hartig, O.: LDQL: A language for linked data queries. In: Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management, Lima, Peru, May 6 - 8, 2015. CEUR Workshop Proceedings, vol. 1378. CEUR-WS.org (2015), *http://ceur-ws.org/Vol-1378/AMW_2015_paper_34.pdf*
8. Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL queries over the Web of Linked Data. In: Proceedings of the 8th International Semantic Web Conference. Springer (2009)
9. Hartig, O., Freytag, J.C.: Foundations of traversal based query execution over Linked Data. In: Proceedings of the 23rd ACM conference on Hypertext and social media (2012)
10. Hartig, O., Özsu, M.T.: Walking without a map: Ranking-based traversal for querying linked data. In: Proceedings of ISWC 2016, Part I. pp. 305–324 (2016)
11. Hartig, O., Pérez, J.: LDQL: A query language for the web of linked data. J. Web Semant. **41**, 9–29 (2016)
12. Prud'hommeaux, E., Bingham, J.: ShapeTrees specification. Editor's draft (May 2020), *https://shapetrees.github.io/specification/spec*
13. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a modular sparql query engine for the web. In: Proceedings of the 17th International Semantic Web Conference (Oct 2018), *https://comunica.github.io/Article-ISWC2018-Resource/*
14. Verborgh, R.: Re-decentralizing the Web, for good this time. In: Linking the World's Information. ACM (2020)
15. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. Journal of Web Semantics **37–38**, 184–206 (2016)
16. Wylot, M., Cudré-Mauroux, P., Groth, P.: Executing provenance-enabled queries over web data. In: Proceedings of the 24th International Conference on World Wide Web (2015)