TPLP **24** (2): 394–420, 2024.

doi:10.1017/S1471068423000054 First published online 25 April 2023

Distributed Subweb Specifications for Traversing the Web[*]

BART BOGAERTS, BAS KETSMAN, YOUNES ZEBOUDJ
Vrije Universiteit Brussel, Belgium
(*e-mails:* Bart.Bogaerts@vub.be, bas.ketsman@vub.be, younes.zeboudj@vub.be)

HEBA AAMER
Universiteit Hasselt, Hasselt, Belgium
(*e-mail:* heba.mohamed@uhasselt.be)

RUBEN TAELMAN and RUBEN VERBORGH
Ghent University – imec – IDLab, Belgium
(*e-mails:* ruben.taelman@ugent.be, ruben.verborgh@ugent.be)

submitted 1 April 2022; revised 2 December 2022; accepted 6 March 2023

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 toward 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. We evaluate our proposal experimentally on a virtual linked web with specifications and indeed observe that not just the data quality but also the efficiency of querying improves.

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

[*] This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme and SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1274521N). Heba Aamer is supported by the Special Research Fund (BOF) (BOF19OWB16).

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 acceptability 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 (Verborgh 2020).

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 (Verborgh *et al.* 2016). 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 (Buil-Aranda *et al.* 2013).

Fortunately, a technique was introduced to query webs of data: *Link Traversal–based Query Processing* (LTQP) (Hartig *et al.* 2009; Hartig 2013a), 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* (Hartig and Freytag 2012). While more data can be considered advantageous, it can also lead to doubts regarding *data quality*, *trustworthiness*, *license compatibility*, or *security* (Taelman and Verborgh 2022). 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, for example, deciding to include parts of a subweb, or to omit it. As such, the data publishers *guide* the data consumer toward relevant data sources. We focus on the theoretical foundations and highlight opportunities for result quality and performance improvements. We implemented our proposal and experimentally validated it on a crafted web of linked data, annotated with subweb specifications in our formalism.

The rest of this paper is structured as follows. In Section 2, we present a motivating use case; afterward, in Section 3 we recall some basic definitions. From our use case, in Section 4, we extract several desired properties. Related work is discussed in light of the use case and the derived desired properties in Section 5. Our theoretical formalism is presented in Section 6, and a concrete web-friendly syntax for it is discussed in Section 7. In Section 8, we investigate to which extent existing link traversal formalisms can "simulate" the behaviour of our formalism. The answer is that even for very simple expressions, such simulations are not possible, thereby illustrating the expressive power of our new formalism. We evaluate the effect of using subweb specifications on performance and on query result quality in Section 9. We end the paper with a discussion and a conclusion.

Publication history. A short version of this paper was presented at the 2021 RuleML+RR conference (Bogaerts *et al.* 2021). This paper extends the short version with proofs and an experimental evaluation.

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 (Verborgh 2020), 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

```
<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.

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 (Hartig and Freytag 2012)), 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 (Cyganiak *et al.* 2014). 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 an *RDF graph*. An *RDF dataset* is a set of tuples $\{\langle n_i, g_i \rangle\}$ such that $n_i \in \mathcal{U}$ 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 $\langle D, \mathit{data}, \mathit{adoc}\rangle$ where D is a set of documents $D \subseteq \mathcal{D}$, data a function from D to $2^{\mathcal{T}}$ such that $\mathit{data}(d)$ 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 (Hartig and Freytag 2012), 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 $W = \langle D, \mathit{data}, \mathit{adoc}\rangle$ and $W' = \langle D', \mathit{data}', \mathit{adoc}'\rangle$. We say that W' is a *subweb* of W if

1. $D' \subseteq D$
2. $\forall d \in D' : \mathit{data}'(d) \subseteq \mathit{data}(d)$
3. $\mathit{adoc}'(u) = \mathit{adoc}(u)$ if $\mathit{adoc}(u) \in D'$ and $\mathit{adoc}'(u)$ is undefined otherwise.

We write $\mathit{subwebs}(W)$ 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 $\mathit{singleton}(d, W)$ to denote the (unique) subweb $\langle \{d\}, \mathit{data}', \mathit{adoc}'\rangle$ of W with $\mathit{data}'(d) = \mathit{data}(d)$.

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 $\langle D', \mathit{data}', \mathit{adoc}'\rangle$ of W with

- $D' = D_{W_1} \cup D_{W_2}$, and
- $\mathit{data}'(d) = \mathit{data}_{W_1}(d) \cup \mathit{data}_{W_2}(d)$ for each $d \in D'$, where, slightly abusing notation, we use $\mathit{data}_{W_i}(d) = \emptyset$ 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, that is, 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. For example, 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 from publishing 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 (Hartig and Freytag 2012), 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 the querying agents* and to allow *querying agents to decide whether or not to include the data from such subwebs.*

Moreover, the fact that published subweb specifications can be used across different applications forms an incentive for document publishers to actually build and publish such specifications. For instance in our running example, if Uma publishes her subweb specficiations for the purpose of the described phone book application, and later she wants to use a different application, for example, a social media app, to access her friends (and their data), she can use the same specifications and simply link to her own data pod.

5 Related work

Web decentralization. To counter the various issues surrounding centralized data management, efforts such as Solid (Verborgh 2020), Mastodon (Zignani *et al.* 2018), and others (Kuhn *et al.* 2021) aim to decentralize data on the Web. While approaches such as Mastodon aim to split up data into several federated instances, Solid takes a more radical approach, where each person can store data in a personal data vault, leading to a wider domain of decentralization. Solid is built on top of a collection of open Web standards (Capadisli *et al.* 2020), which makes it an ecosystem in which decentralized applications can be built. This includes specifications on how data can be exposed through the HTTP protocol (Speicher *et al.* 2015), how to manage authentication (Coburn *et al.* 2022) and authorization (Capadisli 2022; Bosquet 2022), and representing identity (Capadisli and Berners-Lee 2022). Our approach for enabling the publication of subwebs of interest has precedent in the Solid ecosystem, since approaches such as Type Indexes (Turdean 2022) and Shape Trees (Prud'hommeaux and Bingham 2021) already exist that enable data discovery in data vaults by type or data shape. Our approach differs from these approaches in the fact that we use this published information for link *pruning* instead of link *discovery.*

Link traversal–based query processing Over a decade ago, the paradigm of Link Traversal–based Query Processing was introduced (Hartig *et al.* 2009), 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 (Hartig 2013a). For example, the processing order of documents can be changed so that certain documents are *prioritized* (Hartig and Özsu 2016), which allows relevant results to be emitted earlier in an iterative manner (Hartig 2013b), 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 toward relevant (according to the data publishers at hand) documents. LTQP is related to the domain of focused crawlers (Chakrabarti *et al.* 1999; Batsakis *et al.* 2009), which populate a local database by searching for specific topics on Web pages. It is also related to SQL querying on the Web (Konopnicki and Shmueli 1998; Mendelzon *et al.* 1996), which involves querying by attributes or content within Web pages. In contrast to these two related domains, LTQP is based on the RDF data model, which simplifies data integration due to universal semantics.

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 (Hartig and Freytag 2012), 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 simplest reachability criterion is c_{None}, where none of the URIS from the seed documents is followed, such that the query will be executed over the union of triples across the seeds. Query 1 under c_{None} with Document 1 as seed would thus not yield any results. 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 is 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 (Abiteboul *et al.* 2011), 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 their 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 their publication record correctly, whereas another source captures their 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 that she trusts her own profile for her list of contacts, and that she trusts 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 $\sigma : \mathcal{W} \to 2^{\mathcal{U}}$.

Definition 6
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 $W = \langle D, \mathit{data}, \mathit{adoc} \rangle$ and URI u; we extend the notation and also write $f(W,u)$ to denote the subweb $\langle D, \mathit{data}', \mathit{adoc} \rangle$ of W with $\mathit{data}'(d) := f(\mathit{data}(d), u)$ 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 $\sigma(W) = \{o \mid (s, \texttt{foaf:knows}, o) \in \mathit{data}(\mathit{adoc}(s)) \text{ with } s = \texttt{<https://uma.ex/\#me>}\}$ and with a filter that maps (S,u) to $\{(s,p,o) \in S \mid s = u\}$. 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 7
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 $\mathit{adoc}(u)$ 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 knowledge of how to construct the subweb of all data publishers.

Definition 8
A *specification-annotated* WOLD (sa-WOLD in short) is a tuple $\mathbb{W} = \langle W, \boldsymbol{\Theta} \rangle$ consisting of a WOLD $W = \langle D, \mathit{data}, \mathit{adoc} \rangle$ and an associated family $\boldsymbol{\Theta} = (\Theta_d)_{d \in D}$ of subweb specifications.

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

Definition 9
Let $\mathbb{W} = \langle W, \boldsymbol{\Theta} \rangle$ be a sa-WOLD with $W = \langle D, \mathit{data}, \mathit{adoc} \rangle$, and Θ a subweb specification. Then, $[\![\Theta]\!]^{\mathbb{W}}$ denotes the subweb specified by Θ for $\mathbb{W}$,

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

where $(S \textbf{ if } b)$ 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 $\mathit{soi}(d, \mathbb{W}) := \mathit{singleton}(d, W) \cup [\![\Theta_d]\!]^{\mathbb{W}}$.

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 10
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 (Cyganiak *et al.* 2014)). 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.

$$\begin{array}{rcl}
\langle\text{start}\rangle & \models & \textbf{FOLLOW}\ \langle\text{sources}\rangle\ [\textbf{WITH SUBWEBS}]\ [\langle\text{filter}\rangle] \\
\langle\text{sources}\rangle & \models & \langle\text{variables}\rangle\ \textbf{\{}\ \langle\text{GroupGraphPattern}\rangle\ \textbf{\}}\ [\langle\text{recurse}\rangle] \\
\langle\text{variables}\rangle & \models & \textbf{?}\langle\text{VARNAME}\rangle\ \mid\ \textbf{?}\langle\text{VARNAME}\rangle\ \langle\text{variables}\rangle \\
\langle\text{recurse}\rangle & \models & \textbf{RECURSE}\ [\langle\text{INTEGER}\rangle] \\
\langle\text{filter}\rangle & \models & \textbf{INCLUDE}\ \langle\text{ConstructTemplate}\rangle\ [\textbf{WHERE}\ \textbf{\{}\ \langle\text{GroupGraphPattern}\rangle\ \textbf{\}}]
\end{array}$$

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 comprised of 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 $?v_1 \dots ?v_n$ **{** 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, that is, a set of bindings μ (mappings from variables to $\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$).

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

$$\textbf{FOLLOW } ?v_1 \textbf{ \{ } G_1 \textbf{ \} INCLUDE } C \textbf{ WHERE \{ } G_2 \textbf{ \}.}$$

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

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

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 over the topic she indicates. With these subweb specifications, Query 1 produces only Rows 1–3 of Results 1. However, we still include the undesired[1] profile picture from Bob in our results (Row 3). Extending the notion of filter to also allow this is left for future work.

```
<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/*

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 (Hartig 2012), this raises the question: to what 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*: 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 (x_1, x_2, x_3) with result u in the context of a URI u_{ctx} if the following two points hold:

1. there exists $i \in \{1, 2, 3\}$ such that $\ell_i = \_$ and $x_i = u$, and
2. for every $i \in \{1, 2, 3\}$ either $\ell_i = x_i$, or $\ell_i = +$ and $x_i = u_{ctx}$, or $\ell_i = \_$.

[1] Assuming Uma choosing a picture for Bob "overrides" the picture Bob has chosen for himself.

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^*	$\{u\} \cup [\![lpe]\!]^u_W \cup [\![lpe/lpe]\!]^u_W \cup [\![lpe/lpe/lpe]\!]^u_W \cup \ldots$
$[lpe]$	$\{u \mid [\![lpe]\!]^u_W \neq \emptyset\}$

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

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

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

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

that is, 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
Hartig and Pérez (2016) allow one other form of link path expression, where an entire LDQL query is nested 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.[2] 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

$$\langle +, \texttt{foaf:knows}, \_\rangle, /\langle +, \texttt{foaf:knows}, \_\rangle,$$

when evaluated in a given URI u (the context) traverses to u's friends f (as explicated by triples of the form $(u, \texttt{foaf:knows}, f)$ in $adoc(u)$) and subsequently to their friends f_2 (as indicated by triples $(f, \texttt{foaf:knows}, f_2)$ in $adoc(f)$). The final result contains only such friends f_2. In other words, this example expression identifies the documents of friends of friends of a given person.

[2] Notably, this option was also not present in the original work (Hartig 2015).

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

$$\langle lpe_1, P_1\rangle \text{ AND } \langle lpe_2, P_2\rangle,$$

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: P_1 should be evaluated in the documents identified by lpe_1 and P_2 in the documents identified by lpe_2. In this sense, LDQL thus violates 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.

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. For instance, the query

$$\langle lpe_1, (?x, \texttt{foaf:knows}, ?y)\rangle \text{ AND } \langle lpe_2, (?y, \texttt{foaf:mbox}, ?m)\rangle,$$

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 toward 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. That is, 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 11
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 $\mathbb{W} = \langle W, \mathbf{\Theta}\rangle$ a sa-WOLD (with $W = \langle D, data, adoc\rangle$) 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 12
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 $\mathbb{W} = \langle W, \mathbf{\Theta} \rangle$ with $W = \langle D, data, adoc \rangle$ and in which subweb specifications only use triples of the form $(\sigma, \text{true}, id)$ with $\sigma \in \mathcal{S}$ and for each URI u,

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

where

$$docs(S) = \bigcup_{s \in S} singleton(adoc(s), W).$$

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

What this definition states is that the LPE e_{meta}, when evaluated in s identifies precisely all URIS needed to create the subweb of interest of s (including s itself). In case LDQL captures distribution of functions in a certain class $\mathcal{S}$, this means that the knowledge of selectors in $\mathcal{S}$ can be encoded as "meta-triples" in the documents to guide the querying agent toward the relevant data sources. In what follows, we study for some concrete classes whether LDQL can capture distribution.

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, that is, 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:

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

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, for example, 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 (in)expressivity results are then summarized in the following Theorems.

Theorem 1
LDQL captures the distribution of $\mathcal{S}_{const}$.

Proof
We will provide explicit pairs of encoding and meta-expression that capture the distribution.

Consider the class $\mathcal{S}_{const}$; for any (constant) source selector σ in this class, let U be the set of sources defined by it. In this case, we take $enc(\sigma) = \{(a, a, u) \mid u \in U\}$ and $e_{meta} = \langle a, a, \_ \rangle,^*$ with a a fresh URI. The link pattern $\langle a, a, \_ \rangle$, in e_{meta} is used to navigate to the u, while the star ensures that for each u that is found, also their subwebs of interests are included. □

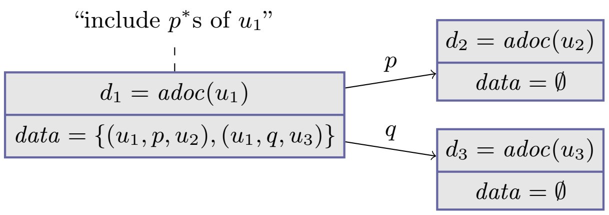

Fig. 1: Example WOLD used in LDQL inexpressivity proof.

Theorem 2
LDQL captures the distribution of $\mathcal{S}_{p^*}$.

Proof
We will provide explicit pairs of encoding and meta-expression that capture the distribution, similar to the proof of Theorem 1, but now for the class $\mathcal{S}_{p^*}$.

We can take $enc(all_{p^*,u}) = \{(a, a, u)\}$ and $e_{meta} = (\langle a, a, \_\rangle, /\langle +, p, \_\rangle,^*)^*$ with a a fresh URI. In this expression e_{meta}, the link pattern $\langle a, a, \_\rangle$, is used to navigate to the u as in the case of $\mathcal{S}_{const}$. Each of these URIS is a source whose ps of ps of... we wish to include; the part $\langle +, p, \_\rangle,^*$ then navigates to all such p^*s. Again, the outermost star ensures that for each u that is found, also their subwebs of interests are included. □

Intuitively, LDQL captured the classes $\mathcal{S}_{const}$ and $\mathcal{S}_{p^*}$, because we wanted the encoding to provide only one piece of information, which is the source to navigate to. Hence, the usage of the meta-expression is to decode that source. In the case of the $\mathcal{S}_*$ distribution, there are two pieces of information that need to be provided by the encoding function *i)* the source to navigate to, and *ii)* the property we need to follow from that source. This can be encoded by some simple *enc* function. Hence, the meta-expression would then need to decode both the source and the property. However, regardless of how the *enc* function will be defined, there is no generic LPE that does this correctly. This is proved in the following theorem.

Theorem 3
LDQL does not capture the distribution of $\mathcal{S}_*$.

Proof
For the sake of contradiction, assume that LDQL captures the distribution of $\mathcal{S}_*$. Then, there exists some pair of encoding and meta-expression that captures $\mathcal{S}_*$. Let this pair be (enc, e_{meta}) with U the set of URIS mentioned in e_{meta}. We can construct a WOLD (see Fig. 1) that uses URIS that do not occur in U in which only one document has a nonempty subweb specification. As shown in Fig. 1, we have two triples (u_1, p, u_2) and (u_1, q, u_3) in d_1 where none of p, q, u_2, or u_3 is in U. Moreover, d_1 has a subweb specification whose source selector is all_{p^*,u_1}. We can also make sure that neither u_2 nor u_3 is mentioned in the triples added by *enc*. This can be safely assumed since *enc* does not depend on the triples found in the W, rather it only depends on the source selector.

Now, if e_{meta} is a correct meta-expression, $[\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$ should evaluate to $\{u_1, u_2\}$. However, any e_{meta} when evaluated at u_1 in our $\mathbb{W}$ would either include u_3 as a selected source or exclude u_2 from the selected sources. Thus, in the rest of the proof we verify this claim.

In order to verify this claim, we first need to show that given any URI u whose authoritative document and subweb specification are empty in some $\mathbb{W}$, then for every LPE e, we have that

$$[\![e]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$$

Intuitively, if we have an empty document without any associated subweb specification, then any LPE evaluated at the source of this document would not result in any sources except for the source of that document, which the context URI. This can be shown by induction on the shape of the LPE e as follows:

- If e is ϵ or e is $[lpe]$, it is clear that $[\![e]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$.
- If e is lp, we have $[\![e]\!]^{u}_{enc(\mathbb{W})} = \emptyset$ since the document is empty and u has no subweb specification.
- If e is lpe_1/lpe_2, it follows by induction that $[\![lpe_1]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$. In case $[\![lpe_1]\!]^{u}_{enc(\mathbb{W})}$ is empty, then $[\![e]\!]^{u}_{enc(\mathbb{W})} = \emptyset \subseteq \{u\}$. Otherwise, $[\![lpe_1]\!]^{u}_{enc(\mathbb{W})} = \{u\}$. In that case, it also follows by induction that $[\![lpe_2]\!]^{u}_{enc(\mathbb{W})} = [\![e]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$.
- If e is $lpe_1 \mid lpe_2$, it follows by induction that both $[\![lpe_1]\!]^{u}_{enc(\mathbb{W})}$ and $[\![lpe_2]\!]^{u}_{enc(\mathbb{W})}$ are subsets of $\{u\}$. Hence, $[\![e]\!]^{u}_{enc(\mathbb{W})} = [\![lpe_1]\!]^{u}_{enc(\mathbb{W})} \cup [\![lpe_2]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$.
- If e is lpe^*, then $[\![e]\!]^{u}_{enc(\mathbb{W})} \subseteq \{u\}$ follows by induction.

Applying this to our example, we see that for any LPE e, it follows that $[\![e]\!]^{u_2}_{enc(\mathbb{W})} \subseteq \{u_2\}$ and $[\![e]\!]^{u_3}_{enc(\mathbb{W})} \subseteq \{u_3\}$. Now we need to show that for every LPE e,

$$u_2 \in [\![e]\!]^{u_2}_{enc(\mathbb{W})} \text{ if and only if } u_3 \in [\![e]\!]^{u_3}_{enc(\mathbb{W})}$$

This can also be verified by induction on the shape of the LPE e, however, it is easily shown from the previous induction since in all the cases where $[\![e]\!]^{u}_{enc(\mathbb{W})}$ did not turn out empty, the result did not depend on the shape of the link patterns used in the expression rather it originally came from the LPE ϵ which is evaluated similarly in u_2 and u_3.

Now, in our example $\mathbb{W}$, we show that for every LPE e_{meta} that does not mention any of the URIS p, q, u_2, or u_3, we have that

$$u_2 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})} \text{ if and only if } u_3 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$$

We verify this claim by induction on the possible shapes of e_{meta} as follows:

- If e_{meta} is ϵ and e_{meta} is $[lpe]$, it is clear that $[\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})} \subseteq \{u_1\}$.
- If e_{meta} is lp, we have three cases to analyze:
  - — lp matching a triple that is added by enc. As mentioned, none of the triples added by $enc(all_{p^*,u_1})$ to d_1 mentions u_2 or u_3. Hence, it is clear that $u_2 \notin [\![lp]\!]^{u_1}_{enc(\mathbb{W})}$ and $u_3 \notin [\![lp]\!]^{u_1}_{enc(\mathbb{W})}$.
  - — lp of the form $\langle l_1, l_2, l_3\rangle$, matching the triple (u_1, p, u_2) in d_1. Since e_{meta} mentions neither p nor u_2, we must have $l_1 \in \{u_1, +, \_\}$ and $l_2 = l_3 = \_$ in order to match this triple. Clearly, each of the three possible link patterns matches the triple (u_1, q, u_3) as well. Thus, $\{u_2, u_3\} \subseteq [\![lp]\!]^{u_1}_{enc(\mathbb{W})}$.
  - — for any other lp, we have $[\![lp]\!]^{u_1}_{enc(\mathbb{W})} = \emptyset$ since there are no other triples in the document.
- If e_{meta} is lpe_1/lpe_2, it follows by induction that $u_2 \in [\![lpe_1]\!]^{u_1}_{enc(\mathbb{W})}$ if and only if $u_3 \in [\![lpe_1]\!]^{u_1}_{enc(\mathbb{W})}$. Accordingly, we have three cases to analyze:

— neither u_1, u_2 nor u_3 belongs to $[\![lpe_1]\!]^{u_1}_{enc(\mathbb{W})}$. In this case, neither u_2 nor u_3 belongs to $[\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$.

— $u_1 \in [\![lpe_1]\!]^{u_1}_{enc(\mathbb{W})}$. By induction, we have that $u_2 \in [\![lpe_2]\!]^{u_1}_{enc(\mathbb{W})}$ if and only if $u_3 \in [\![lpe_2]\!]^{u_1}_{enc(\mathbb{W})}$.

— $\{u_2, u_3\} \subseteq [\![lpe_1]\!]^{u_1}_{enc(\mathbb{W})}$. Thus, the expression lpe_2 will be evaluated once at u_2 and once at u_3. By our previous discussion, we can verify that $u_2 \in [\![lpe_2]\!]^{u_2}_{enc(\mathbb{W})}$ if and only if $u_3 \in [\![lpe_2]\!]^{u_3}_{enc(\mathbb{W})}$.

Thus, in all three cases, $u_2 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$ if and only if $u_3 \in [\![e_{meta}]\!]^{u_2}_{enc(\mathbb{W})}$.

- If e_{meta} is $lpe_1 \mid lpe_2$ or e_{meta} is lpe^*, it follows by induction.

In all cases, we have showed that $u_2 \in [\![e_{meta}]\!]^{u_1}_{enc(\mathbb{W})}$ if and only if $u_3 \in [\![e_{meta}]\!]^{u_2}_{enc(\mathbb{W})}$. Hence, a contradiction. □

9 Experiments

In this section, we present an empirical evaluation of our proposed formalism. The aim of these experiments is to evaluate what the possible gain is when using subweb specifications. Since for the moment, there are no data publishers publishing their subweb specifications, we created a virtual linked web, in which agents do publish such specifications, of which we assume they represent which data sources they trust. On this virtual linked web, we will compare querying the annotated web in our new semantics to querying it under existing reachability semantics. We expect to observe the benefits of using subweb specifications in terms of completeness of the query results, data quality, and performance. As far as data quality and completeness go, we assume here that, since each agent publishes their subweb specifications, the results obtained by querying the subweb-annotated WOLD are considered the "gold standard" (they are the results we wish to obtain since they take every agent's specifications into account). What we want to evaluate is how well existing link traversal can approximate this gold standard: whether query results are missing (incomplete) or whether spurious results are obtained (indicating the use of lower-quality data). We will also evaluate performance: the time needed to traverse the web, as well as the number of traversals required.

Our experiments are run on a virtual linked web constructed from the LDBC SNB (Social Network Benchmark) dataset (Erling *et al.* 2015). The resulting dataset is then fragmented into interlinked datasources resulting in a virtual web of around 301000 datasources with 1.5GB of data. The schema of the virtual linked web is illustrated in Figure 2. To each document type, we attach a subweb specification with the intended meaning that it identifies other data on the web it trusts. We use simple specifications that can naturally arise in real-life scenarios. Most of the defined specifications use filters to only include data that is related to the respective source and ignore other triples that provide information about others. For instance, a person trusts (includes data from) other persons they know, their university, the city where they live, and the company in which they work. The experiments are then run in the annotated virtual linked web. The subweb specifications used in the annotated linked web are listed in Appendix A.

We compare our semantics with the most common reachability semantics criteria (Hartig 2012): c_{All}, c_{None}, and c_{Match}. Our expectation is that c_{All}, which follows all links it

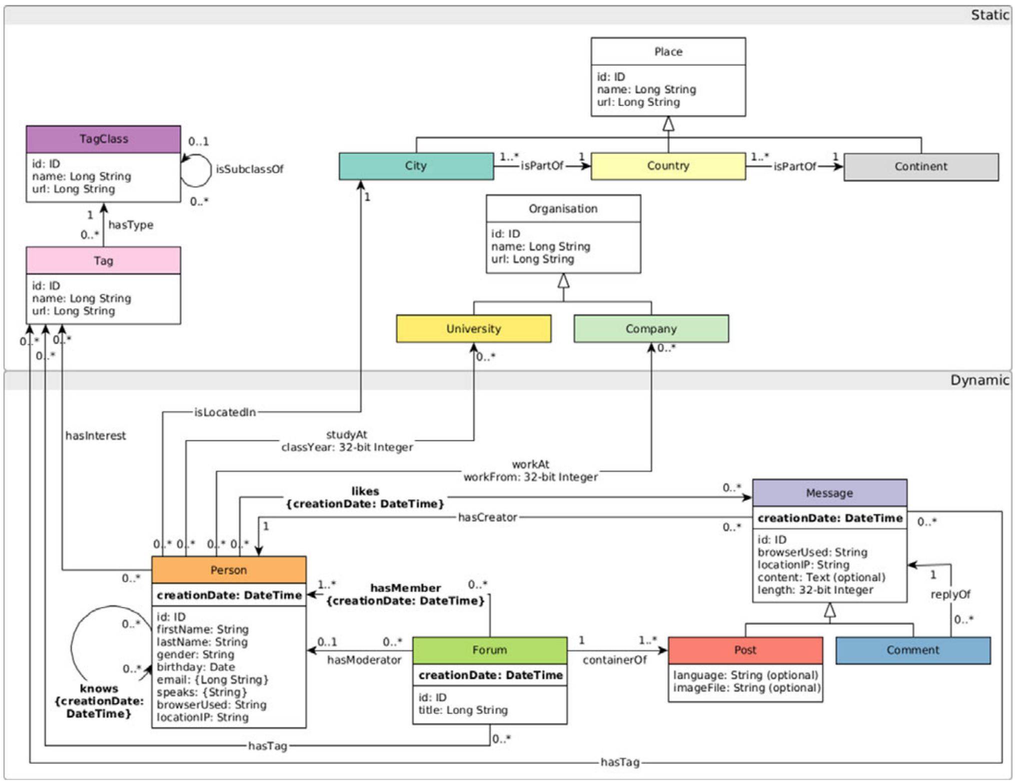

Fig. 2: Schema of the virtual linked web used in the experiments (from *https://www.npmjs.com/package/ldbc-snb-decentralized*).

finds, will be too slow for practical purposes, and generate too many spurious results. For c_{None}, which simply follows no links (and thus only queries the given seed documents), we expect extremely fast querying, but almost no query results, due to the fragmentation of the data. For c_{Match}, which follows all links that somehow "match" the query in question will miss certain query results, but can also generate some spurious results. Indeed, the traversal performed by c_{Match} is only informed by the querying agent, not by the individual users on the web. The *performance* (in terms of runtime and number of links traversed) of c_{Match} compared to subweb-annotated querying is hard to predict. Clearly, the behavior of SWSL itself differs with respect to the filters used in the specifications, which will be elaborated in two of the experimented queries.

The virtual web is hosted locally using a (community[3]) solid server. Each document on the web provides information in the form of a dump file, that is, the RDF dataset which should be downloaded entirely prior to querying. We use the comunica query engine (Taelman *et al.* 2018) as a SPARQL querying engine; the engine uses a caching mechanism that avoids downloading the same file more than once. The experiments are conducted on a personal computer with 16GB of memory and an Intel Core i7 with 2.6 GHz and 6 cores running macOS 12.6.

[3] *https://communitysolidserver.github.io/CommunitySolidServer/docs/*

9.1 Queries

For the evaluation, we used four different queries. The first query (Eval. Query 1) searches for persons' information, it retrieves for each person, the city where they are located along with the country and the continent of this city. The second query (Eval. Query 2) matches persons working in the same company, in which those persons' are retrieved along with the name of the company. In the third query (Eval. Query 3), we test forums members' interests; we want to list members (and moderators) of a forum that have no interest (tag) in common with the forum. The last query (Eval. Query 4) is used to retrieve all persons having an interaction between them; an interaction here is simply a person liking a comment of another; we also list the city where the person who performed the like lives in. For the first two queries, we evaluated our semantics using two different subweb specifications. We refer to the main subweb specification used in all queries as SWSL. The other subweb specification used in evaluating the first query will be referred to as SWSL_1, while SWSL_2 is the other subweb specification used for the second query. The main difference between SWSL_1 and SWSL_2 and SWSL is the filters of the subweb specifications attached to person-type documents. Simply, we allow more triples to be included in the subwebs of person-type documents in the case of SWSL_1 and SWSL_2 than that of SWSL, so the filters used in SWSL are more restricted (see Appendix A for details).

```
SELECT ?person ?country ?cont
WHERE {
  ?person voc:isLocatedIn ?city.
  ?city voc:isPartOf ?country.
  ?country voc:isPartOf ?cont.
}
```

Eval. Query 1: Person's location.

```
SELECT ?person1 ?person2 ?namecomp
WHERE {
  ?person1 voc:workAt ?c1.
  ?c1 voc:hasOrganisation ?comp1.

  ?person2 voc:workAt ?c2.
  ?c2 voc:hasOrganisation ?comp2.

  FILTER(?person1 != ?person2)

  ?comp1 foaf:name ?namecomp.
  ?comp2 foaf:name ?namecomp.
}
```

Eval. Query 2: Same company.

```
SELECT ?forum ?creator
WHERE {
  {?forum voc:hasModerator ?creator.}
  UNION
  {?forum voc:hasMember ?creatorB.
   ?creatorB voc:hasPerson ?creator.}
  ?creator voc:hasInterest ?interest.

  FILTER (NOT EXISTS {
    SELECT ?tagForum WHERE {
      ?forum voc:hasTag ?tagForum.
      FILTER (
        bound(?interest) &&
        ?tagForum = ?interest )}})
}
```

Eval. Query 3: Forum member interests.

```
SELECT ?person ?creator ?city
WHERE {
  ?person voc:isLocatedIn ?city.
  ?person voc:likes ?message.
  ?message voc:hasComment ?comm.
  ?comm voc:hasCreator ?creator.
}
```

Eval. Query 4: Interaction.

Each query is executed 12 times with a different (random) seed URI for each run, and the average is reported. We use the same set of seeds for all strategies to avoid bias related to seeds selection, for instance, forums can have different number of members and persons can have different numbers of posts, comments, or friends.

9.2 Results

The results are displayed in Tables 2, 3, 4 and 5 (one table for each query). In all the experiments, c_{All} and c_{None} behave as predicted, that is, the engine traverses a large number of links when using c_{All}, while no links are traversed when c_{None} is used.

The first query (see Table 2) shows the best traversal performance for c_{Match} to the detriment of query results. Using SWSL$_1$, the engine was able to yield on average 16 results, while c_{Match} was only able to find one of these results. In fact, for each correct result retrieved by SWSL$_1$, a person entity is involved, and since no triple pattern in the query links to other persons, c_{Match} only has the seed document to generate a result in response to the query. On the other hand, SWSL$_1$ was able to traverse to other persons using the subweb specifications of the seed document. It may seem weird to have no results at all from SWSL, but this makes sense since the defined filters are quite strong. In order to be able to include the triple that links the country to its continent, which is of the form `(_:country voc:isPartOf _:continent)`, in the subweb of the current person, this can only be done via the URI of the city where this person is located in. Although this triple is included in the subweb of the city, the filter does not allow the necessary triple to be included in the subweb of the person as it only allows for triples whose subject is the city itself.

The second query shows similar behavior for SWSL (see Table 3). The engine did not yield any results using it. This is due to the structure of the data inside documents and the defined filters. A company, where a person works, is not declared using one triple, instead, a blank node is used as an intermediate entity. That is, we use two triples as in `(_:person voc:workAt _:blankWork)` and `(_:blankWork voc:hasOrganisation _:company)`. This makes the company inaccessible from the subweb of the data publisher, the reason for this is the filter used by the subweb specification published at the person's data set. The filter only allows triples having the person in the subject position, this will only include the triple `(_:person voc:workAt _:blankWork)` which doesn't refer to the actual company. As for SWSL$_2$, on average eleven results are obtained. This change is due to the inclusion of triples whose property is either `voc:hasOrganisation` or `foaf:name` from the subweb of person. What this shows is that some care is required when using excessive filtering. As for c_{Match}, the engine did not yield any results since no triple pattern links for other persons is mentioned in the query.

In the third query (see Table 4), c_{All}, c_{Match}, and SWSL generated the same results. Nevertheless, c_{Match} has to do more link traversal to achieve the querying process, it has also fetched more triples for this. This is mainly due to the fact that c_{Match} does the traversal by using information from the query, the more triples patterns the query has, the more links the engine will have to traverse.

In the last query, c_{Match} produces on average 18 results that seem to be missing from SWSL. This is not true, since all the extra tuples retrieved by c_{Match} are duplicates. Thus, both c_{Match} and SWSL obtained the same *set* of results, but c_{Match} did it in a more efficient manner. The reason for this behavior of SWSL is that defined subweb specifications for persons include a lot of irrelevant data to the query, which are pruned earlier with c_{Match}.

Table 2. *Performance results for (Eval. Query 1). In evaluating c_{All} for this query, the experiment did not yield any results for one of the twelve tested seed sources. The reason was that the number of triples collected from traversing the links phase was so huge, which made the* SPARQL *query engine crash. Hence, the number of triples, the query evaluation time, and the number of results for c_{All} are the averages of the other eleven runs.*

Approach	# traversed links	traversal time	# triples	query evaluation time	# results
SWSL	8187	88 s	5124	1 s	0
SWSL_1	8187	98 s	5550	1 s	16
c_{Match}	4	0 s	703	1 s	1
c_{All}	29,969	474 s	698,126	11 s	1921
c_{None}	0	0 s	343	0 s	0

Table 3. *Performance results for (Eval. Query 2). In evaluating c_{All} for this query, the experiment did not yield any results for three of the twelve tested seed sources. The reason was the same reason as the one mentioned in Table 2. Hence, the number of triples, the query evaluation time, and the number of results for c_{All} are the averages of the other nine runs.*

Approach	# traversed links	traversal time	# triples	Query evaluation time	# results
SWSL	5294	53 s	3195	1 s	0
SWSL_2	5294	55 s	4703	1 s	11
c_{Match}	5	0 s	556	1 s	0
c_{All}	27,251	423 s	597,198	362 s	16372
c_{None}	0	0 s	279	0 s	0

Table 4. *Performance results for (Eval. Query 3).*

Approach	# traversed links	traversal time	# triples	query evaluation time	# results
SWSL	14	1 s	4567	1 s	134
c_{Match}	641	50 s	221,487	29 s	134
c_{All}	32,701	512 s	767735	135 s	134
c_{None}	0	0 s	36	0 s	0

Table 5. *Performance results for (Eval. Query 4).*

Approach	# traversed links	traversal time	# triples	query evaluation time	# results
SWSL	8287	87 s	3881	1 s	35
c_{Match}	95	2 s	1732	1 s	53
c_{All}	29,968	454 s	703,911	2.4 h	56,902
c_{None}	0	0 s	286	0 s	0

10 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 (Wylot *et al.* 2015). In contrast, traditional LTQP cannot make any assumptions of what to encounter behind a link. The work on describing document structures using shapes (Prud'hommeaux and Bingham 2021) 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.

11 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 multisource 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 (Verborgh *et al.* 2016), 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 (Verborgh 2020) 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 (Prud'hommeaux and Bingham 2021).

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 (Taelman *et al.* 2018); 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. This is because we don't expect users to manually create these subweb specifications, but instead are to be created through a user-friendly user interface or automatic learning-based approaches, similar to how today's discovery mechanisms (Turdean 2022; Prud'hommeaux and Bingham 2021) in Solid are managed. Empirical evaluations will shed light on cases where subweb annotated WOLDS and queries result in a viable strategy.

Acknowledgements

We thank the reviewers for their thorough review and comments on the earlier version of this paper.

References

Abiteboul, S., Bienvenu, M., Galland, A. and Antoine, É. 2011. 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*, 293–304. ACM.

Batsakis, S., Petrakis, E. G. and Milios, E. 2009. Improving the performance of focused web crawlers.

Bogaerts, B., Ketsman, B., Zeboudj, Y., Aamer, H., Taelman, R. and Verborgh, R. Link traversal with distributed subweb specifications. In *Rules and Reasoning - 5th International Joint Conference, RuleML+RR 2021, Leuven, Belgium, September 13-15, 2021, Proceedings* 2021, S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, and D. Roman, Eds., vol. 12851. Lecture Notes in Computer Science. Springer, 62–79.

Bosquet, M. 2022. Access control policy (ACP). Editor's draft, Solid.

Buil-Aranda, C., Hogan, A., Umbrich, J. and Vandenbussche, P.-Y. 2013. SPARQL Web-querying infrastructure: Ready for action? In *Proceedings of the 12th International Semantic Web Conference* 2013, vol. 8219. Lecture Notes in Computer Science. Springer, 277–293.

Capadisli, S. 2022. Web access control. Editor's draft, Solid.

Capadisli, S. and Berners-Lee, T. 2022. Solid webid profile. Editor's draft, Solid.

Capadisli, S., Berners-Lee, T., Verborgh, R. and Kjernsmo, K. 2020. Solid protocol. Editor's draft, Solid.

Chakrabarti, S., Van den Berg, M. and Dom, B. 1999. Focused crawling: a new approach to topic-specific web resource discovery.

Coburn, A., Pavlik, e. and Zagidulin, D. 2022. Solid-oidc. Editor's draft, Solid.

Cyganiak, R., Wood, D. and Lanthaler, M. 2014. rdf 1.1: Concepts and abstract syntax. Recommendation, w3c.

Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D. and Boncz, P. 2015. The LDBC social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 619–630.

Hartig, O. 2012. sparql for a Web of Linked Data: semantics and computability. In *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications.*

Hartig, O. 2013a. An overview on execution strategies for Linked Data queries.

Hartig, O. 2013b. 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.*

Hartig, O. 2015. 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*, vol. 1378. CEUR Workshop Proceedings. CEUR-WS.org.

Hartig, O., Bizer, C. and Freytag, J.-C. 2009. Executing sparql queries over the Web of Linked Data. In *Proceedings of the 8th International Semantic Web Conference*. Springer.

Hartig, O. and Freytag, J.-C. 2012. Foundations of traversal based query execution over Linked Data. In *Proceedings of the 23rd ACM Conference on Hypertext and Social Media.*

Hartig, O. and Özsu, M. T. 2016. Walking without a map: Ranking-based traversal for querying linked data. In *Proceedings of ISWC 2016, Part I*, 305–324.

Hartig, O. and Pérez, J. 2016. LDQL: A query language for the web of linked data.

Konopnicki, D. and Shmueli, O. 1998. Information gathering in the world-wide web: The w3ql query language and the w3qs system.

Kuhn, T., Taelman, R., Emonet, V., Antonatos, H., Soiland-Reyes, S. and Dumontier, M. 2021. Semantic micro-contributions with decentralized nanopublication services.

Mendelzon, A. O., Mihaila, G. A. and Milo, T. 1996. Querying the world wide web. In *Fourth International Conference on Parallel and Distributed Information Systems*, 80–91. IEEE.

Prud'hommeaux, E. and Bingham, J. 2021. Shape trees specification. Editor's draft., W3C.

Speicher, S., Arwe, J. and Malhotra, A. 2015. Linked data platform 1.0. Rec., W3C.

Taelman, R., Van Herwegen, J., Vander Sande, M. and Verborgh, R. 2018. Comunica: A modular sparql query engine for the web. In *Proceedings of the 17th International Semantic Web Conference.*

Taelman, R. and Verborgh, R. 2022. A prospective analysis of security vulnerabilities within link traversal-based query processing. In *Proceedings of the 6th International Workshop on Storing, Querying and Benchmarking Knowledge Graphs.*

Turdean, T. 2022. Type indexes. Editor's draft, Solid.

Verborgh, R. 2020. Re-decentralizing the Web, for good this time. In *Linking the World's Information.* ACM.

Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G. and Colpaert, P. 2016. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web.

Wylot, M., Cudré-Mauroux, P. and Groth, P. 2015. Executing provenance-enabled queries over web data. In *Proceedings of the 24th International Conference on World Wide Web.*

Zignani, M., Gaito, S. and Rossi, G. P. 2018. Follow the "Mastodon": Structure and evolution of a decentralized online social network. In *Twelfth International AAAI Conference on Web and Social Media.*

Appendix A Virtual linked web annotation

In this section, we present the subweb specifications attached to the virtual linked web. Table A 1 contains different subweb specifications used in the settings of the experiment, specifically for SWSL. As for the specifications used in SWSL$_1$ and SWSL$_2$, they are simple modifications and they will be mentioned afterward. The symbol <> is used to refer to the agent publishing the subweb. Person-type documents are the ones that have the most information. A person includes information about his city, friends, university, and company. The **WITH SUBWEBS** directive is used by Persons' subweb specifications for friends, cities, and workplaces, this has the effect of also including the subweb specifications of these agents when evaluating the persons' subwebs of interest. The **INCLUDE** directive is used to allow information only about the agents to be included (triples having the agent IRI in the subject position). A forum includes (nonrecursively) the content of its messages (posts and comments), members, and moderators.

Table A 1. *The subweb specifications used in the experiment. The specifications at * are modified in* SWSL$_1$ *and* SWSL$_2$.

Document (# of Instances)	Subweb specification	Information retrieved

Person (1528)	**`FOLLOW`** `?city` **`WITH SUBWEBS`** `{`
	`<> voc:isLocatedIn ?city.`
	`}` **`INCLUDE`** `{ ?city ?p ?o. }`
		City of the Person *

	`FOLLOW` `?person` **`WITH SUBWEBS`** `{`
	`<> voc:knows ?e.`
	`?e voc:hasPerson ?person.`
	`}` **`INCLUDE`** `{ ?person ?p ?o. }`
		Friends of the Person *

	`FOLLOW` `?univ` **`WITH SUBWEBS`** `{`
	`<> voc:studyAt ?u.`
	`?u voc:hasOrganisation ?univ.`
	`}` **`INCLUDE`** `{ ?univ ?p ?o. }`
		University of the Person

	`FOLLOW` `?org` **`WITH SUBWEBS`** `{`
	`<> voc:workAt ?w.`
	`?w voc:hasOrganisation ?org.`
	`}` **`INCLUDE`** `{ ?org ?p ?o. }`
		Company where the Person works

	`FOLLOW` `?comment {`
	`<> voc:likes ?thing.`
	`?thing voc:hasComment ?comment. }`
		Comments of the Person

	`FOLLOW` `?post {`
	`<> voc:likes ?thing.`
	`?thing voc:hasPost ?post. }`
		Posts of the Person

Forum (13750)	**`FOLLOW`** `?message {`
	`<> voc:containerOf ?message. }`
		Posts and Comments of the Forum

	`FOLLOW` `?member {`
	`<> voc:hasMember ?m.`
	`?m voc:hasPerson ?member. }`
		Members of the Forum

	`FOLLOW` `?mod {`
	`<> voc:hasModerator ?mod. }`
		Moderator of the Forum

Table A 1. *Conntinued.*

Document (# of Instances)	Subweb specification	Information retrieved
Comment (151043)	`FOLLOW ?post { <> voc:replyOf ?post. }`	Post (or Comment) of the Comment
	`FOLLOW ?creator { <> voc:hasCreator ?creator. }`	Creator of the Comment
Post (135701)	`FOLLOW ?creator { <> voc:hasCreator ?creator. }`	Creator of the Post
City (1341)	`FOLLOW ?place RECURSE 1 { <> voc:isPartOf ?place. }`	Location hierarchy of the City
Country (111)	`FOLLOW ?continent { <> voc:isPartOf ?continent. }`	Continent of the Country

In SWSL$_1$, we use different specifications for the city and friends of person as follows:

- for the city of the person, the subweb specification is

```
FOLLOW ?city WITH SUBWEBS { <> voc:isLocatedIn ?city.}
INCLUDE { ?s ?p ?o. } WHERE { FILTER ( ?s=?city || ?p=voc:isPartOf ). }
```

- for the friends of the person, the subweb specification is

```
FOLLOW ?person WITH SUBWEBS { <> voc:knows ?e. ?e voc:hasPerson ?person. }
INCLUDE { ?s ?p ?o. } WHERE { FILTER ( ?s=?person || ?p=voc:isPartOf ). }
```

In the schema of the virtual linked web (Figure 2), the predicate `voc:isPartOf` is used to indicate the information "a city `voc:isPartOf` a country" and "a country `voc:isPartOf` a continent", this allows a city document to refer *recursively* to the second information using the **RECURSE** directive, which allows the inclusion of triples of the form `(?city voc:isPartOf ?country)` and `(?country voc:isPartOf ?continent)` in the filtered subweb of `?city`, and hence, these triples are going to be included in the filtered subwebs of persons.

As for the SWSL$_2$, we change only the specification for the friends of person to be as follows:

```
FOLLOW ?person WITH SUBWEBS { <> voc:knows ?e. ?e voc:hasPerson ?person. }
INCLUDE { ?s ?p ?o. } WHERE { FILTER ( ?s=?person || ?p=voc:hasOrganisation || ?p=foaf:name ). }
```

This will allow for knowing the (names of) universities and companies of persons, overcoming the use of an intermediate literal.