Metadata en besturingskenmerken van lagekosteninfrastructuren voor de publicatie van gelinkte data

Metadata and Control Features for Low-Cost Linked Data Publishing Infrastructures

Miel Vander Sande

Promotoren: prof. dr. ing. E. Mannens, prof. dr. ir. R. Verborgh
Proefschrift ingediend tot het behalen van de graad van
Doctor in de industriële wetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. K. De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017 - 2018

UNIVERSITEIT
GENT

ISBN 978-94-6355-090-1
NUR 983
Wettelijk depot: D/2018/10.500/8

Examination board

And the information superhighway showed the average person what some nerd thinks about Star Trek.

– Homer J. Simpson

You tried your best and you failed miserably. The lesson is: Never try.

— Homer J. Simpson

Preface

The last thesis I wrote was at the Polytechnic University of Valencia in Spain, where I was doing a study exchange program to finish my masters. Most pages were written in a dark room without windows, since it was the only place where I could resist the sun, and all the fun. There, it became apparent that, when you have a lot of things going on, writing any kind on dissertation is tough. This would be later confirmed by my mother and her long-winded PhD track. While I was writing, I also took a course called "Internet de nueva generación" (or in English: the new generation internet), because I felt passionate about the subject. It always fascinated me how the internet, and in particular the Web, advanced the way people interact with knowledge and information. As computers evolved, so did the different ways in which Web content could be portrayed, or presented to users.

The first Web pages I visited had white backgrounds, black text, and a few blue links. They were displayed in the early Web browser Netscape, which ran from a floppy disk. Things got a lot more colorful a few years later, when the Yahoo! GeoCities platform allowed teenagers, like me, to build their own (painfully ugly) website with little technical background. I remember how Adobe Flash, with all its graphical power, slowly turned Web pages into sophisticated and animated applications that matched an offline software experience. And how the release of Google Chrome in 2008 marked browsers sufficiently powerful to replace Flash, which made Web applications vendor-independent and back in the hands of open standards such as HTML, CSS and JavaScript. Now, we even run these applications on our phones, where we replaced clicking links by touching, swiping, moving, and even talking. Throughout, the Web has been, and is, the driving force behind true digital innovation, and therefore has a indisputable impact on modern society. The ability to connect, integrate, and repackage information on a global scale, motivated me to tackle one of its many open challenges in my doctoral work.

This new generation internet course introduced me to Semantic Web and Linked Data technology, although I did not actually realize that at the time. The entire class was taught in Spanish, and, because of a collaboration with universities in Barcelona and Madrid, was done over video conferencing with low audio quality. As my understanding of the Spanish language was quite non-existent (which it still is), I spent most of my time in class looking at the figures that were displayed on the slides. Because I sometimes recognized an acronym or a logo, I took comfort in the assumption that I already knew most of the material anyway. Needless to say, if it was not for the graded attendance, I would have flunked miserably. Imagine my surprise when I started recognizing things when reading the introductory material for my new job as a Semantic Web researcher at the Multimedia Lab (now IDLab) of Ghent University and iMinds (now imec).

Now, I know that Linked Data is facilitating technology for the Web applications of today, as they rely, more than ever, on information that is shared over the Web. However, as user expectations of such applications continue to rise, so does the complexity of the technology to share Linked Data. In that process, it is often disregarded that such complexity shift raises the bar for adoption, causing the under-resourced or under-informed organizations to abandon ship. Therefore, this dissertation is about the effects of simplifying these solutions, thus lowering the bar and democratizing the publication of Linked Data.

Since I joined the *Knowledge on Web-Scale (KNoWS)* team in the fall of 2011, I have taken a deep dive into the broad subject of Web, Semantic Web, and Linked Data. Within that domain, our team was able to touch upon an existing vacuum between the theoretical, scientific concepts produced by the research community, and what the industry considers production-ready. As community newcomers, we positioned ourselves as advocates of "stuff that works" to remind everyone that the Web environment is crucial and even indispensable for the dissemination of their work. Here, with the support of iMinds, I could bridge what was a collection of emerging, but still experimental technology with that digital innovation companies were striving for. Our first major team-effort led to *Everything is Connected*: an application that autonomously creates a movie explaining the link between two random concepts using video, images, and text plucked from the Web. That application got us our first conference award, and put us on the map. After that, I pushed for evolving things beyond software engineering and, from scratch, we started implementing better research practice within our team. Over the years, I have seen our methodology strengthen, our papers improve, and our workflows mature, without abandoning our "works in practice" mindset. I can proudly say that this dissertation, and the work by my fellow generation of colleagues, contribute to that effort's first real payoff.

Before, during, and after the writing of this dissertation, many people have contributed—colleague or not, knowingly or unknowingly—who all deserve a proper thanking. First and foremost, my thanks goes out to my entire examination committee, who provided valuable or extensive comments, and, of course, my supervisors. Ruben, thank you for being a great accomplice, inspiration, discussion partner, and co-author. Our collaboration has been more than productive, and calling this doctoral work partly yours is only fair. Together with the team, we made ourselves indisputable in the community, got our Linked Data Fragments work widely known, improved the use of scientific method, and wrote a bunch of high-quality papers. Erik, thank you for keeping the ship afloat, so everyone could work (more or less) worry-free. You, solving my occasional problematic project and enabling me to work half-time for the Krook project, meant a great deal to me. An since I promised some guys on a cycling trip to include a metaphor: just like on the Tourmalet, you stood behind me the entire time.

Further thanks to all KNoWS' former members Tom, Sam C., Davy, Laurens, Dieter D.P., Dieter D.W., Hajar, and current members Pieter C., Anastasia, Ben, Gerald, Martin, Dörthe, Ruben T., Joachim, Pieter H., Brecht, Julian, Cristian, Gayane, and Sven. Working together was always fun and effective, whether it was for a project, a paper, or a conference travel. The same holds for other members of IDLab that have contributed directly or indirectly to my work. A special mention goes out to Femke O., who is always willing to help, guide and review, regardless of the size of her workload—I repaid that in Japanese Ramen, and the tandem Laura and Kristof, who are always willing to help with stupid questions and quests for office supplies. Also a big thanks to all project partners and co-authors that I worked with along the way.

Parts of my research were conducted abroad, which yielded several tools, publications, and events. Therefore, I thank my international companions Karl, Magnus, Dominique, Javier, Laurens R., Olaf, Maria-Esther, Pedro, Jacco, Harish, Lyudmila, and Wouter B. Herbert and Craig, to you I am especially grateful for welcoming me in your institutions overseas. I had two wonderful experiences at ISI-USC in Los Angeles, and the Los Alamos National Laboratory. With the help of your teams, limited time turned into great value.

Thanks are also due to Robby, Kasper, Matthias, Liesbeth, Jole, Sam D., and Wouter D. for joining me in our Krook adventure for imec. This dissertation was not written despite of, but because of our great team; the fun and creativity were a much needed distraction. Although things did not exactly work out the way we wanted to, it was not by a lack of trying.

Thanks too to my mother, for introducing the PhD challenge, my father, for showing endless patience in that regard, and my brother, for setting the family engineering bar high. Also thanks to my friends for all the support, entertainment, company, and the occasional beer.

Now all that is left for me to do is thanking you, my beautiful Sara. Despite being involved so late in the game, you have had a substantial impact on this dissertation, as well as on me. Among many other things, I am grateful for, in order of importance, making sure I did not starve, supporting me in every way possible, and putting my mind of things when needed. I would recommend you to any thesis writer, if you were not already mine.

Miel
Gent, February 6, 2018

Contents

Summary

The information on the Web has grown beyond human processing capabilities at a remarkable rate. Therefore, automated software applications that help users plow through the vast amount of documents (e.g., search engines) have gained presence, with the increasing expectation of being more and more intelligent. Instead of simply returning documents, they should autonomously retrieve, process, and understand Web content to generate precise answers. Hence, because of a machine's flawed understanding of human-readable content, the Semantic Web community started publishing datasets in a structured format, annotated with machine understandable semantics, and with links to each other; all by using the Web's architecture. These efforts led to the deployment of the Web of Linked Data, which now contains more datasets than ever.

Linked Data is most commonly materialized using the Resource Description Framework (RDF), which leverages the global HTTP URI scheme to identify concepts and relationships. To describe knowledge, RDF organizes these URIs in basic relationship statements—called triples—containing three components: a subject, a predicate, and an object. When URIs are shared among datasets, they automatically become interconnected. Using the SPARQL query language, applications can pose complex questions to these RDF collections.

To be properly consumable by applications, data is commonly published using a Web API, of which three are prominent for RDF. The first and foremost option is to publish datasets as *data dumps*, where a dataset is serialized in one or more files and made available for download. The second option is the more granular interface *Linked Data documents*, where a certain URI can be dereferenced to provide more information about that URI in triples. The third option is offering full access to the SPARQL query language over HTTP through a *SPARQL endpoint*. Unfortunately, only a limited fraction of the existing datasets is available in a queryable interface, while existing interfaces suffer from availability issues. Reasons are as economical as they are technical. Because of its fine-granularity, hosting a SPARQL endpoint requires complex and expensive infrastructure. Publishers that cannot afford to host one at high availability, resort to publishing inexpensive data dumps instead. Since these need to be downloaded first, it cannot be considered live querying.

This doctoral work investigates alternative Linked Data interfaces that lead to more sustainability, thereby further democratizing the publication of live queryable Linked Data. Therefore, there is an emphasis on the characteristics that allowed the Web to scale. In particular, the focus is on solutions with minimal server complexity (minimizing the *cost* for data publishers) while still enabling live querying (maximizing the *utility* for Linked Data applications). Such sustainable infrastructure can revive a *virtual integration* strategy. Instead of letting a third-party aggregate and redistribute their data, publishers can offer their Linked Datasets through a query interface of their own and remain in control. Consuming applications integrate data on demand by querying a federation of distributed interfaces, which conforms better to a Web of Linked Data vision.

In order to open up a wider spectrum of possible interfaces, the Linked Data Fragments conceptual model introduced a uniform view on all Linked Data APIs. Each of these APIs publishes a fragment of a certain RDF dataset defined by a *selector*—the condition on which RDF triples are selected—embedded in the request URI. The response to each request is therefore a Linked Data Fragment, consisting of three components: *data*, *metadata*, and *controls*. This model facilitates the identification of gaps in usability, availability, or other requirements that should be addressed, and designing new interface trade-offs that can be evaluated.

As primary novel instance of the Linked Data Fragments model, this thesis defines Triple Pattern Fragments, an interface to RDF triples with low server cost with a triple pattern as selector. This pattern is the basic element of a SPARQL query and is similar to a triple, except that each component can contain a variable that can match one or more triples. Triple Pattern Fragments have all triples of a dataset that match this triple pattern as data, an estimate of the number of triples that match as metadata, and a hypermedia form that enables retrieving any Triple Pattern Fragment of the same dataset as controls. These components are embedded in a self-descriptive response that embeds all information necessary for the client's task. This enables adding features to the Triple Pattern Fragments interface in a backward-compatible fashion; clients discover supported features at run time, which they can use or ignore as they see fit. Supported by Triple Pattern Fragments, this work covers four important capabilities that should be offered to client-side applications in a Web of Linked Data: *(a)* executing complex queries; *(b)* querying multiple sources; *(c)* discovering relevant interfaces; and *(d)* accessing the history of datasets.

To execute complex SPARQL queries over a Triple Pattern Fragments interface, a client architecture and an iterator-based query algorithm are introduced. Based on the cardinality metadata, the algorithm creates a dynamic pipeline that favors the most selective triple pattern to produce incremental results. Evaluation shows that server load is reduced and caching effectiveness increases, leading to lower costs to maintain high server availability. These benefits come at the expense of increased bandwidth and slower, but more stable query execution times. Adding additional approximate membership metadata to cope with the dominance of triple membership requests reduces bandwidth, but introduces too much overhead. It does however enable temporarily allowing imprecise results and retracting these after validation, which speeds up execution for most queries.

The query execution techniques are generalizable to federations of interfaces in a straightforward way; hence supporting virtual integration of several distributed data sources. The Triple Pattern Fragments client is extended with a mediator layer that abstracts a collection of interfaces and eliminates irrelevant sources during query execution. When tested on the public Web, the results show a competitive recall compared to the state-of-the-art SPARQL federation systems and comparable performance for certain queries.

To enable discovering Linked Data interfaces before they can be queried, this thesis also proposes a peer-to-peer discovery method. Using hypermedia and Linked Data principles, distributed interfaces exchange lightweight dataset summaries with an *active* and a *reactive* process. Active discovery scans the dataset for external URIs, dereferences them, and follows the returned links to the other interface's summary. Reactive discovery retrieves the summary of an interface as a reaction to a discovery HTTP request. A client receives hyperlinks to relevant fragments published by other interfaces, which are constructed from the collected summaries, to optimize its source selection process. Hence, the distinction between *query execution* and *federated query execution* fades. In context of evaluation, a quantification of Linked Data interface discovery approaches is introduced to enable proper measurement and comparison with other systems. The experimental results show promise in execution time gain, but poor recall, which points out more intelligent consumption is needed.

Finally, this work covers the problem of query *reproducibility*—a result of ever drifting Web content—by publishing the history of Linked Datasets. Extending the Triple Pattern Fragments interface with a time-dimension enables the same low-cost publishing and querying techniques to be applied for RDF archiving systems. A pragmatic file-based archiving technique is discussed accordingly. The complete toolstack enables analyzing the evolution of facts and synchronizing between sources, which is verified using a digital humanities Use Case for reconstructing institutional history. However, the importance of data provenance information for explaining changes and trusting sources cannot be underestimated.

In conclusion, this doctoral work shows how lightweight interfaces can lower the cost for Linked Data publishers compared to more expressive endpoints, while enabling applications to query the publishers' data with the necessary reliability. Increased attention for smaller datasets from under-resourced data institutions, which are often owners of highly specific and carefully curated data, unlocks more indispensable knowledge. More nuance and demanding less from servers enables doing more with Linked Data; hence, making the Web of Linked Data a sustainable reality.

Samenvatting

Het wereldwijde Web groeit met een ongelofelijke snelheid. De hoeveelheid beschikbare informatie is intussen zo groot dat deze niet langer door mensen kan verwerkt worden. Bijgevolg zijn ook het aantal geautomatiseerde software applicaties, zoals bijvoorbeeld zoekmachines, toegenomen. Zij helpen gebruikers door deze grote hoeveelheid informatie te ploegen. Bovendien wordt verwacht dat deze applicaties steeds intelligenter worden. Zo volstaat het voor de eindgebruiker niet langer dat deze applicaties enkel een lijst van relevante documenten weergeven. Men gaat er namelijk van uit dat deze applicaties autonoom informatie op het Web doorzoeken, verwerken en begrijpen om een precies antwoord op de gestelde vraag te formuleren. Daarnaast bevat het Web veelal inhoud geschreven voor menselijke eindgebruikers wat het voor geautomatiseerde software bemoeilijkt om deze teksten te begrijpen. Om deze redenen begon de Semantisch Web beweging met het publiceren van datasets die op dergelijke manier gestructureerd en geannoteerd zijn dat ze door machines kunnen worden verwerkt. Bovendien zijn deze datasets ook met elkaar gelinkt door middel van de architectuur van het Web. Deze inspanningen hebben geleid tot de implementatie van het Web van gelinkte data dat nu meer datasets bevat dan ooit.

Gelinkte data worden meestal opgeslagen met behulp van het Resource Description Framework (RDF), dat het globale HTTP URI-systeem gebruikt om concepten en relaties te identificeren. Om kennis te beschrijven hanteert RDF een bepaalde formulering — tripletten genaamd — die uit drie componenten bestaat: een onderwerp, een predicaat en een object. Door URIs te delen tussen datasets worden deze automatisch gekoppeld. Met behulp van de SPARQL query taal kunnen applicaties complexe vragen stellen aan deze RDF collecties.

Data worden gewoonlijk gepubliceerd met behulp van een Web Application Programming Interface opdat applicaties ze vlot zouden kunnen gebruiken. Specifiek voor RDF bestaan er drie prominente interfaces. De eerste en populairste mogelijkheid is het publiceren van datasets als *data dumps.* Een dataset wordt in één of meer bestanden geformatteerd en vervolgens online ter beschikking gesteld via download. De tweede mogelijkheid is de meer granulaire

interface *Linked Data documenten.* Door een bepaalde URI te bezoeken, wordt er een document opgevraagd met meer informatie over die URI. Deze informatie is beschreven in RDF tripletten. De derde mogelijkheid is volledige toegang bieden tot de SPARQL query taal via het HTTP protocol aan de hand van een *SPARQL eindpunt.* Helaas hebben er slechts een beperkt aantal van de beschikbare datasets een zo een bevraagbare interface. Bovendien zijn de datasets die wel over een bevraagbare interface beschikken frequent onbeschikbaar. De redenen hiertoe zijn zowel economisch als technisch. Vanwege zijn fijne granulariteit vereist een SPARQL eindpunt een erg complexe en dure infrastructuur. Data verdelers die zich niet kunnen veroorloven om zo'n eindpunt betrouwbaar aan te bieden, verkiezen vaak het publiceren van goedkope data dumps als alternatief. Aangezien deze eerst moeten worden gedownload, kan dit niet als direct bevraagbaar worden beschouwd.

Dit doctoraat onderzoekt alternatieve interfaces voor gelinkte data die leiden tot een meer houdbare situatie en daarmee het publiceren van direct bevraagbare gelinkte data verder democratiseren. Daarom besteedt dit proefschrift extra aandacht aan de karakteristieken die de schaalbaarheid van het Web garanderen. Specifiek wordt er gekeken naar oplossingen met minimale server complexiteit (het minimaliseren van de *kost* voor data verdelers), die tegelijk directe bevraging mogelijk maken (het maximaliseren van de *bruikbaarheid* voor applicaties van gelinkte data). Een dergelijke infrastructuur kan *virtuele integratie* strategieën nieuw leven inblazen. In plaats van hun data te laten aggregeren en herverdelen door een derde partij, bieden verdelers gelinkte datasets aan met hun eigen bevraagbare interface en behouden bovendien op deze manier de controle over hun data. Applicaties die data gebruiken integreren deze op aanvraag door verschillende gedistribueerde interfaces te bevragen, wat beter aansluit bij de visie van een Web van gelinkte data.

Om een breder spectrum van mogelijke interfaces open te breken, introduceerde het conceptueel model Linked Data Fragments een uniforme kijk op alle APIs voor gelinkte data. Elke van deze APIs publiceert een fragment van een bepaalde RDF dataset, gedefinieerd door een *selector* — de voorwaarde waarmee RDF tripletten worden geselecteerd — vervat in de verzoek URI. Het antwoord op elk verzoek is daarom een Linked Data Fragment dat drie componenten bevat: *data*, *metadata*, en *controls.* Dit model ondersteunt de identificatie van opportuniteiten in gebruiksvriendelijkheid, beschikbaarheid of andere vereisten. Het laat ook toe nieuwe afwegingen voor interfaces te ontwerpen die kunnen geëvalueerd worden.

Deze thesis introduceert Triple Pattern Fragments als eerste nieuwe instantie van het Linked Data Fragments model: een interface voor RDF tripletten met lage server kost met een *triplet patroon* als selector. Dit patroon is het basis element van een SPARQL-vraag en is gelijkaardig aan een triplet, behalve dat elk component een variabele kan bevatten waaraan een of meerdere tripletten kunnen voldoen. Een Triple Pattern Fragment heeft als data component alle tripletten van een dataset die aan dit triplet patroon voldoen. Verder heeft het als metadata component een inschatting van het aantal tripletten dat voldoen

en als controls component een hypermedia formulier dat toelaat om elk Triple Pattern Fragment van dezelfde dataset op te halen. Deze componenten zitten vervat in een zelf-beschreven antwoord dat alle nodige informatie bevat voor de taak van de applicatie. Dit laat toe om kenmerken toe te voegen aan de Triple Pattern Fragments interface op een achterwaards compatibele manier. Hierdoor ontdekken applicaties interface kenmerken tijdens de uitvoering die ze, indien nodig, kunnen negeren. Dit werk omvat vier belangrijke mogelijkheden die moeten aangeboden worden aan applicaties op een Web van gelinkte data, ondersteund door Triple Pattern Fragments: *(a)* uitvoeren van complexe bevragingen; *(b)* bevragen van meerdere bronnen; *(c)* relevante interfaces ontdekken; en *(d)* toegang bieden tot de geschiedenis van datasets.

Om complexe SPARQL bevragingen uit te voeren over een Triple Pattern Fragments interface, worden een cliënt architectuur en een iterator-gebaseerde bevragingsalgoritme geïntroduceerd. Op basis van de kardinaliteit metadata creërt het algoritme dynamisch een volgorde van uitvoering die voorrang geeft aan het meest selectieve triplet patroon om iteratieve resultaten te berekenen. De evaluatie toont aan dat de server belasting gereduceerd wordt en effectiviteit van de cache verhoogt, wat tot lagere kosten leidt om hoge server beschikbaarheid te garanderen. Nadelen zijn echter verhoogde bandbreedte en tragere maar stabiele uitvoertijden van bevragingen. Toevoegen van extra benaderde lidmaatschap metadata, om de dominantie van triplet lidmaatschap bevragingen tegen te gaan, vermindert bandbreedte, maar introduceert meer kosten. Het maakt wel het tijdelijk toelaten van inaccurate resultaten mogelijk, wat voor veel bevragingen de uitvoering versnelt.

De technieken voor het uitvoeren van bevragingen zijn direct uitbreidbaar naar collecties van interfaces, dus ook virtuele integratie wordt ondersteund. De cliënt wordt uitgebreid met een mediator laag die een collectie van interfaces abstraheert en irrelevante bronnen elimineert tijdens de uitvoering. De resultaten van testen op het publieke Web tonen een competitieve volledigheid in vergelijking met de modernste SPARQL federatie systemen en een vergelijkbare uitvoertijd voor sommige bevragingen.

Deze thesis stelt een gelijke-naar-gelijke ontdekkingsmethode voor interfaces tot gelinkte data. Door gebruik te maken van hypermedia en de principes van gelinkte data kunnen gedistribueerde interfaces lichte dataset samenvattingen uitwisselen met een *actief* en *reactief* proces. Actieve ontdekking overloopt de dataset voor externe URIs, vraagt ze op en volgt de links naar de samenvatting van de andere interface. Reactieve ontdekking vraagt de samenvatting van een andere interface op als reactie op een ontdekking-HTTP verzoek. Een cliënt ontvangt hyperlinks naar relevante fragmenten die gepubliceerd zijn door andere interfaces, geconstrueerd uit de verzamelde samenvattingen, om het bron selectieproces te optimaliseren. Hierdoor vervaagt het onderscheid tussen het *uitvoeren van bevragingen* en het *uitvoeren van bevragingen over interface collecties.* In context van evaluatie is een kwantificatie van gelinkte data ontdekkingsmethodes geïntroduceerd om correcte metingen en vergelijkingen tussen systemen mogelijk te maken. De resultaten van de experimenten tonen winst in uitvoeringstijd, maar tegelijk zijn de antwoorden onvolledig, wat wijst op een nood voor meer intelligente consumptie.

Ten slotte kaarten we in deze thesis het probleem van *reproduceerbaarheid* aan. Bevragingen over gedistribueerde bronnen zijn moeilijk om correct te reproduceren, omdat inhoud van het Web temporeel uit elkaar drijft. Dit kan worden opgelost door ook de geschiedenis van gelinkte datasets te publiceren. De Triple Pattern Fragments interface uitbreiden met een tijdsdimensie laat aan een lage kost data te publiceren en RDF archieven te bevragen. Aansluitend bespreekt dit doctoraat een pragmatische techniek voor archivering. De totale verzameling oplossingen laat toe om de evolutie van feiten te analyseren en tussen bronnen te synchroniseren. Dit tonen we aan met een casus uit de digitale geesteswetenschappen over de reconstructie van een institutionele geschiedenis. Het belang van informatie over de herkomst van data kan echter niet worden onderschat om veranderingen te verklaren en bronnen te vertrouwen.

Dit doctoraat toont aan dat lichtere interfaces de kosten voor verdelers van gelinkte data kunnen verlagen ten opzichte van meer expressieve eindpunten, terwijl applicaties de gepubliceerde data met de nodige betrouwbaarheid kunnen bevragen. Verhoogde aandacht voor kleinere datasets van instituties met weinig middelen, die vaak eigenaars zijn van zeer specifieke en voorzichtig gecureerde data, opent meer onmisbare kennis. Meer nuance en minder van servers eisen maakt het mogelijk om meer uit gelinkte data te halen, wat het Web van gelinkte data een houdbare realiteit maakt.

List of Figures

List of Tables

I can't promise I'll try, but I'll try to try.

— *Bart Simpson*

Introduction 1

Linked Data enables machines to autonomously retrieve, process, and understand Web content. However, access to these data seems to be problematic: only a limited number of datasets is available through a queryable interface, while existing interfaces fail to offer a reliable service. The reasons seem to be as economical as they are technical. This dissertation pinpoints the current caveats in Linked Data publishing that explain the apparent absence of datasets in live queryable form. Consequently, it investigates new Web interfaces and appropriate consumption techniques to improve the status quo.

This may sound familiar. Someone tells you about a movie starring a famous actor, but somehow, you cannot remember his name. After fruitless attempts to search your memory, you eventually surrender and accept that you just can't recall. Such predicaments are quite common, and yet, it sounds almost nostalgic nowadays. With the *World Wide Web*—the internet's most popular application—at our disposal, a few interactions with a search engine on your smartphone prevent it from ever happening. Moreover, depending on your age, you might not even know what I'm talking about.

This is called "tip of the tongue" phenomenon. Usually, somebody suddenly does recall hours later.

Today, the Web has outgrown itself from a simple document exchange system to a common memory, complimenting the traditional media as primary source of information. In its twenty-five years of existence, over four billion Web pages have been created [1, 2]; that is an incredible accomplishment. What is truly remarkable though, is that we access this unseen volume of information through a single system, a global distributed machine that simply works. Thanks to the simplicity of its protocol (i.e., HTTP) and the flexibility of the content formats (e.g., HTML), it was allowed to scale and to sustain [3].

With the growing quantities of information, the way we interact with the Web evolved—along with our expectations. To aid *surfers* in finding information faster, automated software applications, like search engines, have gained more and more presence. Since then, there has been an increasing appeal for these services to be *intelligent* [4]: autonomously retrieving, processing, and understanding Web content to generate precise answers, rather than returning a list of matching documents. Accordingly, all the big Web companies have introduced their own digital assistant by now: Google Now, Apple's Siri, Amazon's Echo, and Microsoft's Cortana. Given the aforementioned scenario, they would give you a straight answer to the question "What actor starred in the movie X"; or at least try to.

For over a decade of research, preparing the Web for such autonomous computer programs—so-called *intelligent agents* [5, 4]—has been the priority of the Semantic Web community. This effort involves dealing with an important caveat: computers cannot understand and interpret Web content, they need data. This led to the development of a *Web of Linked Data* [6] by using the Web's architecture to publish datasets in a structured format, annotated with machine understandable semantics, and with links to each other. Now, more datasets exist as Linked Data than ever before [7].

As of May 2016, the LODstats project [8] indicates over 150 billion Linked Data facts distributed over 9,960 datasets.

Despite these efforts, however, a significant amount of these Linked Data are not easily accessible, nor reliably live queryable, on the Web. Easy access is mostly a matter of interoperability: agents should be able to consume data in a uniform way without much prior knowledge or development effort. Live queryable enables agents to ask questions directly to the data, which limits processing costs and guarantees freshness of results. Essentially, it takes well-designed Web APIs to publish Linked Data in such a manner.

Yet, the majority of datasets is simply published as downloadable dumps [9], which cannot be considered a live queryable form—the dataset needs to be downloaded first. Some datasets are published using a live queryable API, but suffer from frequent downtime [10]. To offer a Quality of Service (QoS) to their users, however, applications require a reliable data source. Unfortunately, the complexity or demand by this QoS often exceeds the foreseen, considered, or guaranteed service by data publishers. Hence, consuming present Linked Data requires applications to do much of the heavy processing, or deal with the unavailability; little alternative APIs exist. This limits building the envisioned intelligent agents to the happy few: those who can afford extensive dataset crawling, large-scale data integration, high-volume centralized data storage, and a complex query infrastructure [11]. Hence, it is hardly a surprise that Google, Apple, Amazon and Microsoft are currently pioneering in this area.

Throughout the years, my research team and I developed a vision that tackles this issue by addressing the current sparse choice in Web APIs to publish Linked Data. In the current status-quo, publishers choose between deploying an expressive query infrastructure, which is expensive to host reliably (i.e., not suffering frequent downtimes), or hosting inexpensive, yet not directly queryable, data dumps. Hence, we must reconsider our options regarding Web-scale publication of Linked Data. Between the two extremes mentioned above lies a whole spectrum of possible Web interfaces, which has remained largely unexplored. This led to *Linked Data Fragments*, a conceptual model to methodically analyze the benefits and drawbacks an interface brings for clients and servers.

During the course of my doctoral work, I have defined and evaluated alternative Web interfaces, supported by the Linked Data Fragments model. In particular, these interfaces aim at minimal server complexity (minimizing the cost for data publishers) while still enabling live querying (maximizing the utility for applications). This enables more (under-resourced) parties to host an affordable, live queryable, and reliable service. On this premise, my work covers four important capabilities that interfaces should offer client-side applications:

1. executing expressive queries (i.e., able to join different facts);
2. querying more than one source;
3. discovering interfaces on the Web that are relevant to a query;
4. ensuring query reproducibility through accessing prior dataset versions.

This research initially defines an interface restricted in the granularity of the queries it accepts. Each capability corresponds with a series of transparent, interchangeable, and discoverable features that enhance this interface. These features either modify the navigation or the metadata it provides, of which I measured and analyzed the effect in a practical Web context; hence, extra attention was given to network communication, caching infrastructure, server load, and client numbers to test real-world feasibility.

I believe the outcomes of my research further democratize Linked Data publishing, conceiving more live queryable Linked Data. The essence is that, by enabling more nuance and demanding less from servers, you can get more done, hence, making the Web of Linked Data a sustainable reality.

1.1 Research questions

To facilitate *reliable* Linked Data applications, the publishing Web interface should be sufficiently useful for clients, while being affordable to host with high availability. Hence, the first research question of this thesis is rather general:

Research Question 1: Can a Web interface for Linked Datasets designed for low server cost enable complex live querying for clients?

To answer this question, an interface should combine benefits from both expressive interfaces and the non-expressive data dumps. Therefore, we compromise on the query granularity of the interface to make queries more predictable for servers. A coarse-grained interface limits the expressiveness of the individual queries, thus lowering the processor (CPU) cost to answer them? From a pure performance perspective, restricted expressiveness is a major limitation to query execution. However, we argue that, despite sacrificing performance for lower server load, the query response time will still be acceptable for real-world scenarios; that is, executing queries used in actual Web applications, in contrast to the overly complex synthetic queries built to stress test systems. This leads to the following hypothesis:

Hypothesis 1: A client can execute real-world queries over a restricted Linked Data Web interface with a response time below four seconds and a lower server CPU load than more expressive interfaces.

A maximum waiting time of four seconds without additional feedback to the user is generally considered acceptable [12]

For an acceptable response time, a client needs to process queries with a minimum level of efficiency. In database literature, this process is optimized using various types of statistical metadata [13]. However, given the distributed Web environment, where client and server usually do not share the same machine, we pose our second research question:

Research Question 2: To what extent can interface metadata enhance complex query evaluation in a practical Web setting?

For each request, each type of metadata has to be shipped from the server to the client. We selected two suitable metadata types from literature: *(a) cardinality*, an estimation of the number of results; and *(b) approximate membership*, a compact representation of the result set. In addition to query response time, we investigate their applicability in a practical setting, i.e., the impact on network traffic, client load and server load. Hence, we state the following hypothesis:

Hypothesis 2: Extra metadata reduces at least one third of the network traffic required by clients to answer a query.

As mentioned at the beginning, this work aims to facilitate a Web of Linked Data. Hence, besides querying a single interface, we should also enable querying multiple semantically integrated interfaces. Therefore, we pose our third research question:

Research Question 3: Can restricted low-cost Linked Data Web interfaces form an efficient architecture for query evaluation over a federation of interfaces?

A low-cost server-side interface establishes a more balanced division of labor between client and server. The server provides the data and some limited query processing, while the client is responsible for the global query execution. This architecture is very *native* to existing federation frameworks (e.g., ANAPSID [14], FedX [14], and SPLENDID [15]), where servers are not aware of each other and, hence, require a client to integrate the partial results. Thus, our hypothesis is the following:

Hypothesis 3: A client can evaluate queries over a federation of low-cost interfaces on a public network with a performance similar to the state-of-the-art federation frameworks ANAPSID, FedX, and SPLENDID.

Being part of a distributed system, Linked Data interfaces are scattered across the Web. Hence, before we can query a federation of interfaces, we have to discover them first [16], which leads to our fourth research question:

Research Question 4: How can we effectively discover distributed Linked Data interfaces?

From a piece of data, Linked Data allows discovering other related data through its links, but this does not ensure efficient querying yet. Hence, we investigate whether its principles can be applied to interfaces, leading to the following hypothesis:

Hypothesis 4: A client can effectively discover distributed interfaces by relying solely on Linked Data principles.

When querying multiple interfaces, the data they publish might start drifting in time. This requires temporal synchronization between interfaces. However, up to this point, access to prior versions of Linked Datasets has not been considered. Thus, we pose our fifth and final research question:

Research Question 5: Can low-cost interfaces improve access to prior versions of Linked Datasets?

As our low-cost interfaces are of lower complexity, they could also simplify the implementation of existing Web versioning strategies [17]. This should enable clients to navigate from version to version transparently. Hence, we formulate our final hypothesis:

Hypothesis 5: Clients can query published prior versions of Linked Datasets without specifying the exact version.

1.2 Outline

This dissertation reports my findings and developments in eight chapters. It clusters peer-reviewed work from three journal and three conference publications, which are listed at the end of each chapter. Following this introduction, my research is covered by the following chapters:

Chapter 2 – The Web of Linked Data introduces background knowledge on Linked Data and its relation to the Semantic Web. I introduce the different standards used throughout this dissertation and cover the current Linked Data publishing interfaces.

Chapter 3 – Sustainable APIs for Linked Data publishing explains what Web interfaces Linked Data publishers use, and what their current pitfalls are. Subsequently, I discuss important characteristics for Web APIs, introduce the Linked Data Fragments model accordingly, and describe how to explore new Linked Data interfaces.

Chapter 4 – Query Execution illustrates how clients can execute queries on the public Web using a low-cost interface, covering both Research Question 1 and Research Question 2. In particular, I highlight different kinds of metadata an interface can provide to aid the client in this process.

Chapter 5 – Federation of interfaces argues to revive virtual integration as publishing strategy where each publisher has its own interface and integration happens on the client. Hence, we extend query execution over multiple low-cost interfaces by introducing a mediator layer, thereby covering Research Question 3.

Chapter 6 – Discovering interfaces discusses my work on discovering Linked Data interfaces using hypermedia and how it assists query clients to select relevant sources. We enable interfaces to discover each other using HTTP concepts, exchange summaries of the data they publish, and inform clients about their peers. Thereby, this chapter investigates Research Question 4.

Chapter 7 – Accessing history addresses distributed interfaces drifting over time by introducing a few archiving techniques and a straightforward solution to publish them on the Web. This enables synchronizing sources and executing queries over time, resolving Research Question 5.

Chapter 8 – Conclusions evaluates the posed research questions and compiles the lessons learned from the preceding chapters. Also, it looks at the way forward: which fundaments did we lay down and what challenges are left? Based on my research outcome, I question some established aspects in Linked Data querying on the public Web, and suggest a couple of research directions.

References

[1] April Netcraft. *Web Server Survey.* 2004.

[2] Antal Van den Bosch, Toine Bogers, and Maurice De Kunder. "Estimating search engine index size variability: a 9-year longitudinal study." In: *Scientometrics* 107.2 (2016), pp. 839–856.

[3] Roy T Fielding and Richard N Taylor. "Principled design of the modern Web architecture." In: *ACM Transactions on Internet Technology (TOIT)* 2.2 (2002), pp. 115–150.

[4] James Hendler. "Is there an intelligent agent in your future?" In: *Nature* 11 (1999).

[5] Michael Wooldridge and Nicholas R Jennings. "Intelligent agents: Theory and practice." In: *The knowledge engineering review* 10.2 (1995), pp. 115–152.

[6] Tom Heath and Christian Bizer. "Linked data: Evolving the web into a global data space." In: *Synthesis lectures on the Semantic Web: theory and technology* 1.1 (2011), pp. 1–136.

[7] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. "Adoption of the Linked Data Best Practices in Different Topical Domains." English. In: *International Semantic Web Conference.* 2014, pp. 245–260. ISBN: 978-3-319-11963-2. DOI: 10.1007/978-3-319-11964-9_16.

[8] *LODStats.* AKSW. 2016. URL: http://stats.lod2.eu/ (visited on 05/01/2016).

[9] Ivan Ermilov, Michael Martin, Jens Lehmann, and Sören Auer. "Linked Open Data Statistics: Collection and Exploitation." In: *Knowledge Engineering and the Semantic Web.* Ed. by Pavel Klinov and Dmitry Mouromtsev. Vol. 394. Communications in Computer and Information Science. Springer, 2013, pp. 242–249. ISBN: 978-3-642-41359-9. DOI: 10.1007/978-3-642-41360-5_19.

[10] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. "SPARQLWeb-Querying Infrastructure: Ready for Action?" In: *The 12th International Semantic Web Conference.* Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz. Nov. 2013.

[11] Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. "Linked Data-as-a-Service: The Semantic Web Redeployed." In: *European Semantic Web Conference.* Ed. by Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann. Springer International Publishing, June 2015, pp. 471–487.

[12] Fiona Fui-Hoon Nah. "A study on tolerable waiting time: how long are web users willing to wait?" In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163.

[13] Surajit Chaudhuri. "An overview of query optimization in relational systems." In: *The 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems.* ACM, 1998, pp. 34–43.

[14] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. "ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints." In: *The Semantic Web – ISWC 2011.* Ed. by Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist. Vol. 7031. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 18–34. ISBN: 978-3-642-25072-9.

[15] Olaf Görlitz and Steffen Staab. "SPLENDID: SPARQL Endpoint Federation Exploiting VoID Descriptions." In: *The 2nd International Workshop on Consuming Linked Data.* Ed. by Olaf Hartig, Andreas Harth, and Juan Sequeda. Bonn, Germany, Oct. 2011. URL: http://uni-koblenz.de/~goerlitz/publications/GoerlitzAndStaab%5C%5C_COLD2011.pdf.

[16] André Freitas, Edward Curry, João Gabriel Oliveira, and Sean O'Riain. "Querying Heterogeneous Datasets on the Linked Data Web: Challenges, Approaches, and Trends." In: *IEEE Internet Computing* 16 (1 2012), pp. 24–33.

[17] Herbert Van de Sompel, Michael Nelson, and Robert Sanderson. *HTTP Framework for Time-Based Access to Resource States – Memento.* Request For Comments 7089. Internet Engineering Task Force, Dec. 2013. URL: https://tools.ietf.org/rfc/rfc7089.

Can't you people take the law into your own hands? I mean, we can't be policing the entire city!

— *Chief Wiggum*

The Web of Linked Data 2

At the rate the Web is growing, an attractive vision is having intelligent computer programs to help us plow through these vast volumes of information. However, because of their flawed understanding of natural languages, a precondition is a massive deployment of structured, machine-understandable, and interconnected data on the Web. This chapter discusses the different practices, standards, and technologies used to deploy such Web of Linked Data.

When the Web came about, it was primarily designed for human consumption [1]. People were enabled to put text online quick and painless, containing links to other documents that might be relevant—so-called *hypertext* [2, 3]. This paradigm swiftly turned the World Wide Web into a revolution. Over 4.7 billion pages [4, 5] currently in existence bring about the world's largest source of information. Unsurprisingly, those Web documents are hardly written by hand anymore, but generated from structured data contained in databases.

A recent google search for Venus returns pages about the planet, the brand and tennis player Venus Williams.

Yet, the rapidly growing volume of available documents uncovered a fallacy in the resulting dominance of natural language: machines struggle to understand or interpret Web content. By publishing the raw structured data itself, however, a large potential is uncovered. First, machines are enabled to handle tasks more intelligently. For instance, current keyword-based search engines can return a high number of documents (high recall), but are not able to give you an exact answer (low precision). Furthermore, they are receptive to ambiguous terms. Access to raw data can increase the accuracy of answers and disambiguation. Second, publishers are encouraged to *break down data silos*, i.e., positioning a dataset in a context broader than its own. When raw data is no longer only available to its owner, it can be reused by third-parties. Furthermore, the data can be connected to records or facts residing in external databases, eventually creating a global distributed database with uniform access.

To realize both facets, the W3C community project *Linking Open Data* [6] proposed to construct a *Web of Linked Data* [7]. That the goals are ambitious, is clear from this statement by Heath and Bizer [8]:

> Just as the World Wide Web has revolutionized the way we connect and consume documents, so can it revolutionize the way we discover, access, integrate, and use data.

Linked Data proposes to expose, share, and interconnect previously independent datasets via the Web's architecture. Starting from a single piece of data, any person or machine is able to find other related data through hyperlinks. However, opposed to documents, these links also connect arbitrary things and concepts.

2.1 Architecture of the Web

Before we can fully understand Linked Data and how its possible publication methods work, we need a basic understanding of the Web's architecture. It was designed to scale globally and this turned out to be the key factor for its popularity. In the following, we explain its core components, the REST constraints and how they are used to construct APIs.

2.1.1 Core components and architectural properties

The internet and the Web are commonly confused. In reality, the internet is a global network of interconnected computers which runs several other applications as well, like email or instant messaging.

In essence, the Web is a *distributed hypermedia* information system that runs on top of the internet. With distributed, we mean that content does not have to be stored on a single machine, but can be shared by several, stored in different locations. With hypermedia, we mean a generalized concept of hypertext, where not only text is interconnected by links, but also other forms of media, e.g., video and images.

At its conception, Berners-Lee, Cailliau, and Groff [9] defined the core components for the identification, transport, and representation of a *web resource*: anything that can be obtained from or identified on the Web. This originally referred to static files and documents, but evolved over time to cover basically "any information that can be named: a document or image, a temporal service (e.g. 'today's weather in Los Angeles'), a collection of other resources, a non-virtual object (e.g. a person), and so on" [10].

The characterization of the Web's architecture, however, happened at a later stage by Fielding [10], who captured the architectural principles of large-scale distributed hypermedia systems in his conceptual framework *Representational*

State Transfer (REST). Its biggest contribution is protecting *scalability* by introducing a total of six constraints, which allow expansion without negatively impacting its actors. We discuss the most relevant constraints below.

Client-Server

The Web is an instance of the *client-server* model: a distributed application where a central server provides data to a number of connected client machines. They communicate using the standardized HTTP protocol [11], which defines a request–response messaging pattern demonstrated in Figure 2.1. Thereby, a Web client performs an action on a Web resource by sending an HTTP request to a Web server, which contains the following components:

- an **HTTP Verb** indicating the desired operation on the resource, which is either `GET` (request the resource representation), `HEAD` (request the resource without representation), `POST` (perform an action), `PUT` (add the resource), or `DELETE` (remove the resource);
- a **Uniform Resource Locator (URL)** to identify the subject resource. A typical URL could have the form http://www.example.com/index.html, which includes the protocol (HTTP), a hostname (`www.example.com`), and a path that identifies the resource in that domain (`index.html`);
- a number of **Request Header fields**[1] to supply certain parameters such as the originating host, what file format is desired, or the length of the message body;
- an optional **Message Body** containing additional content.

In return, the server send an HTTP response back with these components:

- a **Status Code** indicating whether the request succeeded (ranges 1XX and 2XX), failed (ranges 4XX and 5XX) or further action is required (ranges 3XX);
- a number of **Response Header fields** containing additional information from the server, such as caching information or the file format of the message body;
- an optional **Message Body** containing the *representation* of the requested resource.

Header fields are colon separated (e.g., `Host:` www.example.com) and are defined in the specification. However, custom headers are also possible.

Resources can be represented in a variety of formats, the most well-known undoubtedly being the Hypertext Markup Language (HTML). It is the standard markup language for creating human-consumable hypertext pages. Through clickable links and submission forms, these pages can contain actionable hypermedia to other resources, identified by their URL. Besides HTML, many other representations exist in accordance to the type of resource. Examples are structured data formats such as JSON and XML, or media formats such as JPEG.

1. https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.3

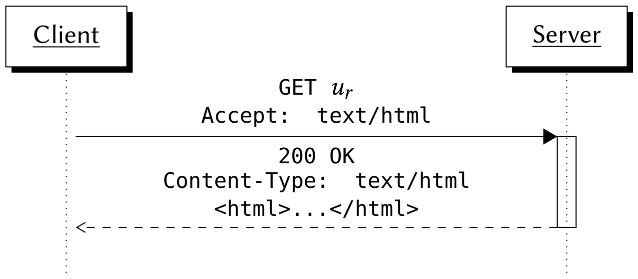


Figure 2.1: A client requests a resource identified by URL u_r in a HTML representation using HTTP. The server responds successfully with the HTML page.

Stateless and Cacheable

To enable a server to scale over many clients, the HTTP request-response cycle should be *stateless*. Note that this applies to *application state*, i.e., the point in the client's session with the server, and not to the *resource state*. A server does store the value of the resource that is reflected in the representation and can be modified using the right HTTP verb.

The server, however, should not store any state about the client's application on the server side, which is the only way to scale to millions of concurrent users. No context is preserved between requests, thus a request contains all necessary information to process it. A client can interact with a server without prior knowledge by passing (parts of) the applications state to the server as needed.

Their statelessness allows resources to be cached and reused by others. Fielding [10] describes its importance as such:

> The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions.

On the Web, clients and intermediaries can cache responses. Therefore, responses must implicitly or explicitly define themselves as cacheable to not prevent clients from reusing stale or inappropriate data in response to further requests. This is enabled by the HTTP protocol through the response headers `Expires` and `Etag` that tells a cache—who intercepts the request—how long it should store the response.

A request can also explicitly state it should not be cached with `Cache-Control: no-cache`. To indicate when the resource last changed, the server also adds a `Last-Modified` header to the response.

Caching is implemented in stages: a request is routed through several caches before it reaches the server. Based on the caching headers in the request, any of these caches can validly intercept the request and already return a cached response. Figure 2.2 illustrates the different caches that can be applied to the interaction in Figure 2.1. We identify two common types of caches: *(a)* a *local cache* implemented on the client for resources it already requested, and *(b)* an *HTTP cache* implemented on the server or a proxy server for resources others already requested.

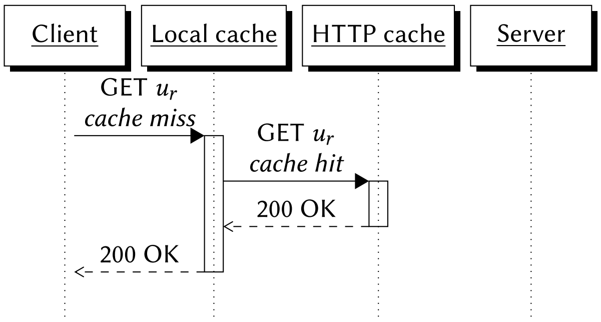


Figure 2.2: An HTTP response can be cached by the client and several intermediaries, unloading the server for all reusable requests.

Uniform interface

The uniform interface is the core differentiating feature of the REST architecture. Besides simplifying and decoupling the architecture, it enables clients and server to evolve independently [10]. This constraint is implemented on the Web through four sub constraints.

Resource identification in requests. Resources are identified by Uniform Resource Identifiers (URI), a generalization of the HTTP URL, so they can become the subject of an HTTP request (e.g., as target of a hyperlink). Note that a URI can only identify one resource, but a resource can be identified by multiple URIs.

Resource manipulation through representations. A client can only assess a resource as zero or more representations (each available in a suitable data format, for instance HTML, or in case of an image, JPEG) and manipulate a resource through zero or more representations. They represent the state of a resource as a byte-string, encoded in a format indicated by the *internet media type* or MIME type, e.g., `text/html` (document) or `image/jpeg` (image). Both the server and client can choose the MIME type at will, which is included in the HTTP headers (e.g., `Accept` and `Content-Type`).

Self-descriptive messages. All exchanged messages include all the information necessary to be processed and cannot rely on other messages. Statelessness is one aspect, but also the well-understood HTTP verbs contribute.

Hypermedia As The Engine Of Application State (HATEOAS). After a client has accessed a resource, the response should include server-provided links that offer all the available actions and resources it needs to carry on. If interaction continues, succeeding responses will contain hyperlinks to different actions that are currently available. Thereby, navigation between resources happens through *hypermedia controls* such as links or forms, which *drive* the application state—like an engine. As a result, client and server require no *out-of-band* information to interact, such as documentation.

2.1.2 Web Application Programming Interfaces

The REST constraints are a recommended architectural style for building a "Web API" (Web Application Programming Interface) or "Web service". In the broadest sense, a Web API is any application that communicates through the HTTP protocol [12]. Regular websites can be conceived of as APIs for humans, where the interface is determined by the HTML pages offered by a server. In the stricter sense, the term "Web API" mostly refers to interfaces that have been designed for an automated, machine-based consumption of Web content. Such an interface can range from providing access to a single resource to an entire dynamic network of interconnected resources.

Instead of REST, some APIs follow a remote-procedure calling (RPC) style, in which HTTP simply acts as a tunneling mechanism for method invocation. RPC also applies a client–server interaction with a request-response mechanism. However, similar to object oriented programming, the API is centered around *actions* a client performs on the server. This often leads to a certain *contract* between client and server, sacrificing scalability for developer comfort.

In contrast to RPC, REST captures the design of the human Web: we browse the Web by clicking links and forms (respectively `<a>` and `<form>` in HTML). Therefore, with REST Web APIs, machines similarly use such hypermedia controls to navigate from one resource to another [13, 14]. A *hypermedia control* is a declarative construct that informs clients of a hypermedia interface of possible application and/or session state changes in their interaction with a server, and explains them how to effectuate such changes. The benefit of REST APIs is that, like websites, they are self-describing: once the basic mechanisms (HTML, XML, ...) are implemented, no external documentation is necessary to browse and consume the API or website.

2.2 Linked Open Data

Initiated by Tim Berners-Lee's call for "raw data now" [15], there is an ongoing effort to make data a first-class citizen on the Web. Situated in a larger, societal context, this premise became the goal of the Open Data movement [15]. They pursue making certain data freely available on the Web for anyone to use or redistribute, without any legal restrictions. Enabling anybody to build applications with your data, addresses the creativity, input, and workload of a huge community. As a result, the return can be way larger than a single company can handle, while the investment is a lot less.

Around the world, many governments have started designing and implementing Open Data strategies in light of transparency and economic potential [16]. To aid them in their deployment, Berners-Lee [17] defined a five star scheme:

- ⋆ put data on the Web with an open license[2];
- ⋆⋆ make the data structured (e.g., spreadsheets instead of an image);
- ⋆ ⋆ ⋆ have it in a non-proprietary format (e.g., csv instead of Excel);
- ⋆ ⋆ ⋆⋆ use URIs to identify things, so people can point at it, and return something meaningful when requested;
- ⋆ ⋆ ⋆ ⋆ ⋆ link your data to other data to provide context.

The fifth star urges data owners to adopt a Linked Data approach, which Bizer, Heath, and Berners-Lee [7] define as "a set of best practices for publishing and connecting structured data on the Web". Applying these practices, and their supporting technology, can integrate distributed datasets semantically, and thus, completely break the data silos in the process. When published under an open license, the contraction Linked Open Data (LOD) is used instead. Every few years, the Linking Open Data project [6] publishes the *LOD-cloud* [18, 19]: a diagram symbolizing all datasets available on the Web and the existing links between them. Figure 2.3 displays the 1,139 available datasets in 2017, illustrating the variety of participating domains. Coming from only twelve datasets ten years ago, we can consider the interest in Linked Data far from over.

Linked Data is most commonly materialized using the Resource Description Framework (RDF) [20], which entails the use of basic relationship statements and the global HTTP URI scheme to identify resources. Linking is achieved by reusing the same URI to refer to the same resource, or by expressing equivalence between different URIs that identify the same resource [21]. In Section 2.3.1, RDF is explained in more detail as part of the Semantic Web.

2.3 Relation to the Semantic Web

When there is talk about Linked Data, the term Semantic Web is never far off. In essence, the Semantic Web is the *vision* to make the current Web more *machine-understandable* [22]. This effort entails "putting data on the Web in a form that machines can naturally understand, or converting it to that form." [23] With additional *semantics*, machines are able to execute sophisticated tasks that go beyond parsing and transforming. For instance, if a machine parses an HTML page about train routes, it needs a programmer to interpret the content and implement how to propose travel routes to a user. When this page is semantically annotated, any program that understands the semantics of transport can apply route planning autonomously. Hence, what used to be software programs, can now evolve into *intelligent agents* [24].

Demonstrated by the rise of digital assistants (as mentioned in Chapter 1), the presence of such intelligent agents is slowly becoming a reality. With the

2. Popular open licenses are the Open Data Commons Public Domain Dedication and License (PDDL), the Open Data Commons Attribution License (ODC-by), and the Creative Commons Zero (CC0).

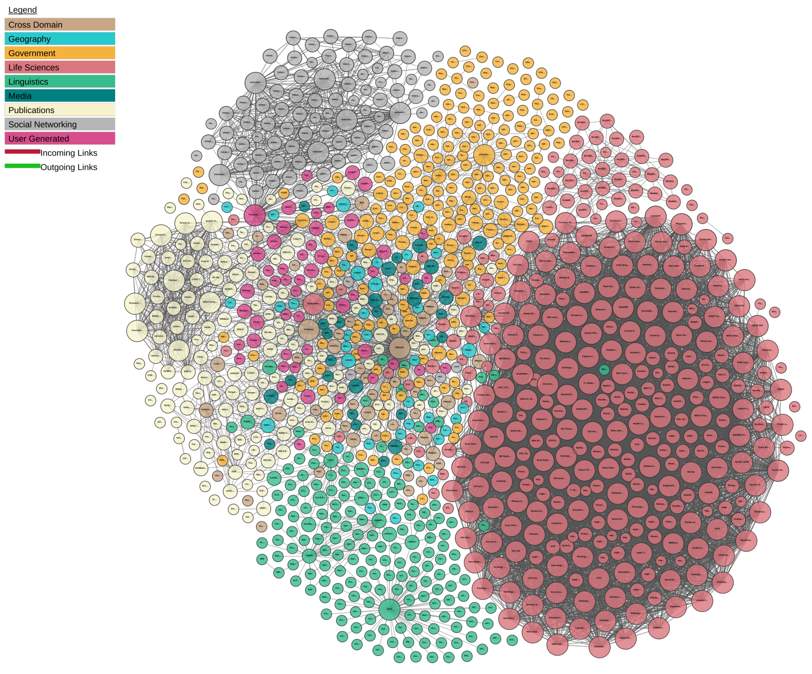


Figure 2.3: Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

Web growing at accelerating rate, it became generally accepted that agents are a necessity in order deal with the increasing complexity [25]. These agents, however, go beyond that what can be witnessed today. After instructing your personal agent with a task, it is able to browse the Web, collect the relevant data, connect pieces of information, and even communicate with other intelligent agents. Important here is the established Artificial Intelligence concept of *knowledge representation* [26]. Knowledge is represented in a formal domain model or *ontology*, so that machines can make complex decisions autonomously, often through means of *reasoning*. From the facts expressed in data, new knowledge can be derived by applying descriptive logic [27].

This ability to roam the Semantic Web as a *global dataspace* is the merit of Linked Data [8]. In an attempt to clarify their relationship, Tim Berners-Lee described Linked Open Data as "the Semantic Web done right"[3] and the Semantic Web as "a web of data that can be processed directly or indirectly by machines." [23], making both terms co-occur ever since. During the unfolding research efforts, the Semantic Web community was sidetracked from the Web aspect, and mainly focused on getting the knowledge management right. The

3. http://www.w3.org/2008/Talks/0617-lod-tbl

Linking Open Data project therefore attempts to reinstate some important neglected aspects, such as actually "building a global Web of machine-readable data" [7].

Gaining knowledge from the Semantic Web, for example by means of reasoning, operates under an *open world* assumption [29, 30]. In a Web context, this means that when a statement is absent in a certain document, it is impossible to determine whether it is false or nonexistent. Hence, we are never in possession of the whole truth, as "anyone can say anything about anything" [20].

Berners-Lee, Hendler, and Lassila [22] conclude that we simply have to "accept that paradoxes and unanswerable questions are a price that must be paid to achieve versatility" (e.g., contradicting statements can coexist in Wikidata [28]).

As this makes absolute sense in a global data space, the open world assumption is reflected in the Semantic Web technology stack. This stack refers to a collection of standards, led by the World Wide Web Consortium (W3C), and remains under active development. In the remainder of this section, we discuss the prominent ones:

Resource Description Framework (RDF) a framework to represent structured knowledge in triples, consisting of a subject, predicate, and object [20].

SPARQL Protocol and RDF Query Language (SPARQL) a combination of a language to query RDF data in stores [31] and a communication protocol [32] on top of HTTP to use the SPARQL language remotely.

Web Ontology Language (OWL) a domain modeling language to represent complex knowledge about things and the relations between them [33]. It is computational logic-based, so reasoners can exploit the expressed knowledge.

2.3.1 Representing Linked Data in the Resource Description Framework

The Resource Description Framework (RDF) [20] is a graph-based representation of information on the Web. Adopted as a W3C recommendation in 1999, RDF was initially launched as a metadata exchange format [34] to establish "a vendor-neutral and operating-system-independent system of metadata" [35]. At its launch, RDF was heavily tied to the hierarchical XML for syntax so existing tools could be reused. Both were developed around the same period and strived to improve interoperability; but in contrast to XML, RDF saw little uptake. It was not until the Semantic Web vision gained widespread believe, that the standard was successfully revived. Today, RDF is used more generally to describe concepts or model information contained in Web resources. It provides a model and syntaxes to implement and represent Linked Data [20].

In RDF, data is expressed as *triples*. A triple is a statement composed of a *subject*, *predicate*, and *object* and can hence be considered as a simple sentence. For instance, the statement below describes the relationship between Umberto Eco and one of his works:

```
"The Name of the Rose"'s author is Umberto Eco.
```

In this sentence, we can distill the subject (*The Name of the Rose*), predicate (*has author*), and object (*Umberto Eco*). In RDF, each of these components is in fact a Web resource and is therefore identified by a URI. Thus, the sentence above is modeled as follows:

```
<http://dbpedia.org/resource/The_Name_of_the_Rose>
      <http://dbpedia.org/ontology/author>
            <http://dbpedia.org/resource/Umberto_Eco>.
```

By using URIs, each concept is uniquely identified on the Web, which enables unambiguous referencing to, for instance, books, authors and the relationship "has author". Sharing these among systems achieves interoperability and interconnection between data.

There are several serialization formats standardized by W3C to represent RDF data:

RDF/XML: a verbose XML syntax to smoothen transition in the early days [36]. It has been largely succeeded by more efficiently processable syntaxes such as Turtle.

N-Triples: the most straightforward syntax that encodes triples line-based in plain text [37]. Triples and URIs are written and repeated in full, making it quite verbose.

Turtle: a lean superset of N-triples that uses shorthands and URI prefixes to avoid verbosity by repetition [38]. The user-friendly Turtle is the most common today, and is used in this thesis to represent RDF.

Some serializations also support graph statements, also known as *quads*. A set of RDF triples—an RDF graph—can be identified by a single URI, which can be perceived as an additional fourth triple component.

N-Quads: a superset of N-Triples adding a fourth graph element per line.

Trig: a superset of Turtle that adds a bracket-based syntax for graphs.

JSON-LD: a JSON representation that approaches RDF from a more Linked Data and Web API perspective [39]. Regular JSON constructs are annotated with RDF terms in a separate predefined *context*.

An RDF *graph* can represent more complex knowledge. For instance, the statements below enhance our prior example statement. Note that we use Turtle syntax to write triples, which should, just like N-Triples, use angular brackets (<>) to indicate URIs. However, for brevity, Turtle allows to abbreviate URIs with prefixes, like so:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
dbr:The_Name_of_the_Rose  dbo:author dbr:Umberto_Eco.
dbr:The_Name_of_the_Rose  rdf:type   dbo:Book.
dbr:Umberto_Eco  dbo:birthDate  "1932-01-05"^^xsd:date.
```

The above triples state that *something* identified by `dbr:The_Name_of_the_Rose` is of type `dbo:Book` and has a relationship identified by `dbo:author` with *something* identified by `dbr:Umberto_Eco`. In turn, the thing identified by `dbr:Umberto_Eco` is related to 5 January 1932 with the predicate `dbo:birthDate`. However, by visiting the URIs and applying the machine-readable semantics captured by the schema—in this case the DBpedia ontology prefixed by `dbo:`—a software application can interpret that a *book The Name of the Rose* is authored by *Umberto Eco*, who was born on 5 January 1932.

The triple on the first line is identical to the triple we discussed earlier.

Note that in RDF, the object of a triple can have a literal value as object instead of a URI, which is the case in the last line. Literals are used to represent basic values like strings, numbers, and, in this case, dates, and can have an optional datatype (also represented by a URI). In many cases, the use of literals is unavoidable. However, for proper linking, the use of URIs is preferred for reusable concepts, so they are referenceable.

Frequently, the XSD datatypes [40] are used which also occur in XML files.

A subject or object can also contain a *blank node*, which is a resource in the RDF graph not identified by URI or literal.

2.3.2 Querying with SPARQL

Knowledge can be derived from Linked Data by querying it. For RDF collections, this is commonly done using the SPARQL1.1 Query Language and Protocol, a W3C specification [31]. The acronym "SPARQL" refers to two distinct concepts:

1. **the SPARQL language**: a query syntax and algebra to formalize questions, and;
2. **the SPARQL protocol**: a HTTP interface for clients to request the execution of SPARQL queries by a server.

The latter can be considered a Web interface, so we discuss it later in this section in conjunction with other Linked Data interfaces. The former defines a query language with an SQL-like syntax used to select and retrieve RDF data. In the SPARQL language, the basic construct is a *basic graph pattern (BGP)*. A BGP is able to match a number of triples in a dataset, and is composed of one or more triple patterns. Just like an RDF triple, a triple pattern also contains a subject, predicate, and object, except, each term can be a wildcard. For instance, a triple pattern aimed at selecting all works by Umberto Eco is expressed as follows:

```
?work dbo:author dbr:Umberto_Eco.
```

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name
WHERE {
  ?work dbo:author dbr:Umberto_Eco.
  ?work dbo:name ?name.
}
```

Listing 2.1: This SPARQL query selects all works by Umberto Eco.

Finding triples that match this pattern is the task of the SPARQL query processor. For this example, every triple in the dataset that has `dbo:author` as predicate and `dbr:Umberto_Eco` as object will match the above query. With a BGP, we can add a collection of connected triples enclosed in brackets; for instance, to also return the label:

```
{
 ?work dbo:author dbr:Umberto_Eco.
 ?work dbo:name ?name.
}
```

The first triple pattern yields a set of URIs, each identifying a *work* authored by Umberto Eco. The second triple pattern yields the label associated with each of those URIs. That is, from the description of the work, the object of the triple that has the work's URI as subject and the `dbo:name` as predicate is chosen.

In addition to triple patterns, a BGP can contain `FILTER` clauses, which pose conditional constraints on the BGP's solutions. For instance, to yield only the work "The Name of the Rose", the values that match `?name` can be filtered with a regular expression:

```
{
 ?work dbo:author dbr:Umberto_Eco.
 ?work dbo:name ?name.
 FILTER regex(?name, "The Name of the Rose")
}
```

In total, there exist four query types that define the operation performed on BGPs:

`SELECT`: returns a result set of variables bounded to the matching triples;
`CONSTRUCT`: creates new RDF triples by substituting values from the matching triples in triple templates;
`DESCRIBE`: returns an implementation-specific description of the matching resources;
`ASK`: returns a Boolean indicating whether the pattern matches or not.

Given the prior example, Listing 2.1 shows a complete SPARQL `SELECT` query.

The SPARQL language offers many assisting constructs as well, which turn it into a very expressive language. These include optional parts (i.e., `OPTIONAL`), shorthands (e.g., `PREFIX`), composition (e.g., `UNION` and `INTERSECT`), negation (e.g., `FILTER NOT EXISTS` and `MINUS`), and aggregation (e.g., `COUNT` and `DISTINCT`).

2.3.3 Modeling domains with the Web Ontology Language

In the Semantic Web domain, the terms *Vocabulary* and *Ontology* are often used synonymously, referring to a domain model of defined concepts and their relationships. The result can vary from very formal, i.e., defining inference rules, to a more loose approach, i.e., defining flexible data schemas.

For RDF, ontologies are often expressed in RDF Schema (RDFS) [41] and its successor the Web Ontology Language (OWL) [33]. As these are not directly relevant to this dissertation, we will not discuss them in more detail. Note however that many of the URIs in predicates and objects originate from reused ontologies and eventually materialize the semantics for a machine.

2.4 Extract-Transform-Load workflows for RDF data

An organization's RDF collection is often not continuous, but is generated periodically from the current data collection with an Extract-Transform-Load (ETL) process. Data can reside in heterogeneous sources, which need more complex transformations first in order to be unified [42].

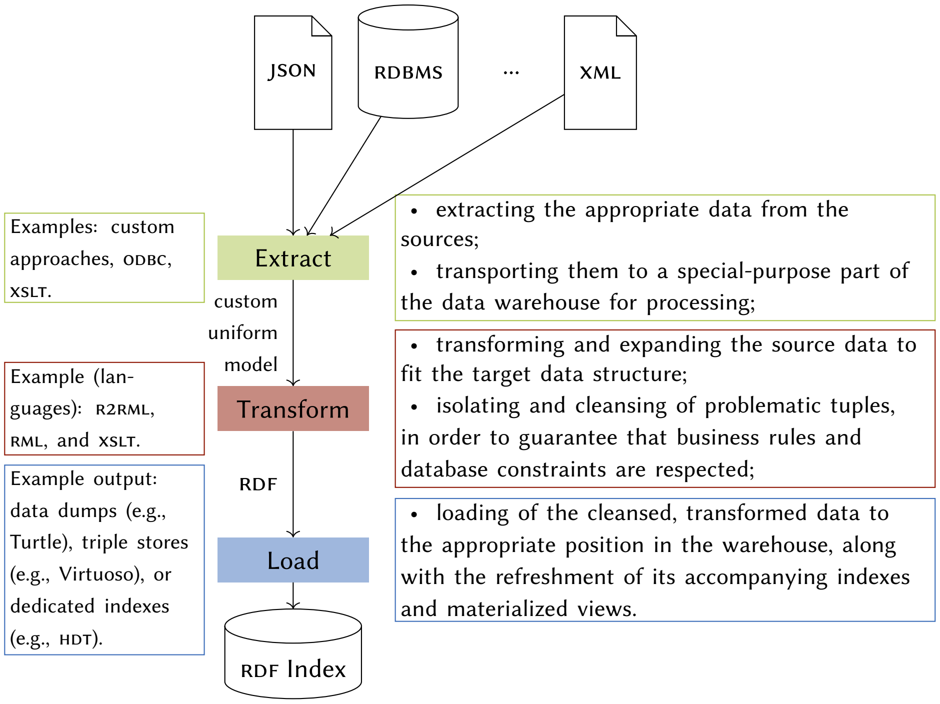

JSON
RDBMS
...
XML
Examples: custom approaches, ODBC, XSLT.
Extract
custom uniform model
Example (languages): R2RML, RML, and XSLT.
Transform
RDF
Example output: data dumps (e.g., Turtle), triple stores (e.g., Virtuoso), or dedicated indexes (e.g., HDT).
Load
RDF Index
• extracting the appropriate data from the sources;
• transporting them to a special-purpose part of the data warehouse for processing;
• transforming and expanding the source data to fit the target data structure;
• isolating and cleansing of problematic tuples, in order to guarantee that business rules and database constraints are respected;
• loading of the cleansed, transformed data to the appropriate position in the warehouse, along with the refreshment of its accompanying indexes and materialized views.

Figure 2.4: Adequate tools exist for RDF to execute the five responsibilities of Extract-Transform-Load workflows defined by Vassiliadis [43].

2.4.1 A three-step workflow

The ETL process originates from data warehousing, where one or more disparate sources are integrated in a central repository. In total, the three-step workflow "Extract, Transform, and Load", handles five responsibilities defined by Vassiliadis [43], as shown in Figure 2.4. Adequate RDF tools exist for each responsibility, which are briefly discussed below.

The *Extract* step is implementation-specific and deals with the heterogeneity of data. Many different file formats are read into a main uniform data model so the RDF ETL tool can process it. This model is often kept in memory and should be sufficiently flexible to not hinder the transformation to a graph structure. Most tools target structured [44] or unstructured [45] data. There is little reuse except for established extraction methods such as XSLT [46] or ODBC [47].

An alternative to ETL is Ontology-based Data access (OBDA), where semantics-based operations on the data are interpreted at runtime; hence, the RDF representation only exists virtually. Most RDF mapping languages can be used for both OBDA and ETL. Popular examples are Ontop [48], Morph [49], or Ultrawrap [50].

The *Transform* step compels RDF publishers to adapt the ETL workflow [51]. Extracted heterogeneous data can conveniently *(a)* be translated into a RDF graph structure, *(b)* adopt or construct resource URIs, and *(c)* be annotated with schema information. This process became known as RDF mapping [52], and is now supported by many ETL tools [53]. To increase the uniformity of the approaches and make mapping definitions reusable, a few mapping languages came about to interpret relational databases as RDF. A well-known example of such language is the W3C Recommendation R2RML [54]. In order to also support formats other than tabular source formats (e.g., spreadsheets and relational databases) [55], Dimou et al. [56] developed RML: a superset of R2RML that adds support for heterogeneous and hierarchical sources.

The *Load* step prepares the RDF for publication or analysis, very often through search and querying. Its target is a data dump in one of the serialization formats or an RDF index, which enables efficient lookups of URIs, patterns or triples. A combination of different indexes with a SPARQL processor is called an RDF database or triplestore. Common examples are Virtuoso [57] or AllegroGraph [58]. These are complex systems that consist of many modules, requiring large-scale infrastructure when the data size is large. However, also more lightweight, dedicated indexes exist. A prominent example is Header-Dictionary-Triples (HDT): a compressed self-indexed binary RDF format [59].

2.4.2 Header-Dictionary-Triples (HDT)

Designed with exchange in mind, HDT targets the ever increasing data volumes by dealing with the redundancy, verbosity, and inefficient machine processability custom to RDF representations. With HDT, an RDF dataset is encapsulated into a single file, consisting of three components:

- a *Header* with metadata about the dataset for discovery and as entry point;
- a *Dictionary* to encode all URIs and literals to avoid redundancies;
- a *Triples* encoding scheme which both compresses and indexes for search operations.

The result is a highly compressed read-only binary archive, which reduces the original dataset size up to 15 times [59]. For big semantic data management systems, HDT offers 25% storage space compared to state-of-the-art RDF indexes, while still competing in query performance [60].

The *Triples* component enables very efficient search and browse operations. Triple pattern lookups can be performed fast without having to decompress any data, keeping storage and memory usage within acceptable bounds. In addition, it can estimate the cardinality of such patterns efficiently, which is useful for optimizing query planning over HDT files.

An HDT file is *immutable*; that is, only read operations can be performed after its creation. This is a considerable limitation in comparison to triple stores or other writable indexes. However, for some use cases like archiving, the inability to change can be considered an advantage. The HDT generation process is also slow and memory-intensive, which may or may not be a problem depending on the use-case. In the use-cases mentioned in this dissertation, though, HDT makes an excellent storage candidate due to its query capabilities and limited size. Therefore, it plays an important role in the described experiments later on.

2.5 Publishing Linked Data

Just like any form of data, Linked Data is published using a Web API, so applications can consume it properly. In the following, we discuss the most prominent used interfaces.

2.5.1 Data dump

A *data dump* is the most straightforward way to publish a Linked Dataset: one or more files contain all triples of the dataset, serialized in one of the RDF representations, e.g., Turtle or N-Triples. These files—possibly compressed into an archive—are uploaded to a fileserver and made available for download. From a REST perspective, the entire dataset is the Web resource, which is identified by its download URL. Common data dumps are the DBpedia datasets[4], Freebase[5] Geonames[6], or the Virtual International Authority File (VIAF)[7]. More datasets can be found at https://old.datahub.io/dataset?res_format=RDF.

4. http://wiki.dbpedia.org/develop/datasets
5. https://developers.google.com/freebase
6. http://www.geonames.org/export
7. http://viaf.org/viaf/data

For publishers, data dumps are a low-complexity solution, as file servers are fairly simple and inexpensive. However, they cannot be considered a solution for *live queryable* data: before SPARQL queries can be executed, clients must download (or stream) the dataset in its entirety over HTTP, ingest them in a SPARQL-aware system, and process the query locally. On the one hand, this offers universality: clients can process datasets as they see fit and ingest them in an access point of choice, optimized for the kind of task they want to perform—even beyond SPARQL queries. On the other hand, if the files are large, there is a significant bandwidth and client processing cost involved. Depending on the use case and total dataset size, this also means the cost for clients to use the data is high, and thus possibly out of reach for a significant number of consumers. Therefore, data dumps mostly suit data-intensive tasks. For a client that only needs a few triples, downloading the entire dump to retrieve these triples is very inefficient. Finally, if (part of) the data becomes outdated, the client has to restart downloading and processing entirely.

Some initiatives have aimed to facilitate the usage of data dumps. For instance, the LOD Laundromat [61] harvests data dumps from the Web in various RDF formats, cleans up quality issues relating to the serialization, and republishes the dumps in standards-conform formats at their own location (including new download URI).

2.5.2 SPARQL endpoints

As mentioned earlier, we can make a much more fine-grained selection of RDF data with the SPARQL query language [31] through highly specific and flexible custom queries. For a client, the easiest way to execute such a query is to ship it to a server that hosts the corresponding dataset. This interaction is standardized in the SPARQL protocol [32]: a client sends SPARQL queries through a specific HTTP interface, the SPARQL *endpoint*, and the server, attempts to execute these queries and responds with their results. Such an interface is supported by many triple stores, such as Virtuoso [57] and Jena TDB [62]. Popular public running instances are DBpedia (http://dbpedia.org/sparql), the BioPortal SPARQL portal (http://sparql.bioontology.org), and the scolary data portal DBLP (http://dblp.rkbexplorer.com/sparql).

```
SELECT DISTINCT ?t
WHERE {
  ?s rdf:type ?t
}
```

Listing 2.2: This SPARQL query selects all distinct classes in the dataset.

A SPARQL endpoint is identified by the standardized base URL which exposes a single resource `/sparql`, to which a query is appended to construct the HTTP request URI. For example, the result of executing the query in Listing 2.2 on the DBpedia dataset endpoint, is available as the following resource:

```
dbpedia.org/sparql?query=SELECT+DISTINCT+%3Ft+%7B+%3Fs+rdf%3Atype+%3Ft+%7D
```

From a REST perspective, this URI identifies the Web resource, which is the result set (represented in one of the SPARQL result formats). In a sense, SPARQL endpoints therefore expose an infinite number of REST resources, which are determined by the clients' choice of a query.

While enabling clients to send arbitrary SPARQL queries leads to low bandwidth consumption and low client cost, the processing of individual requests is potentially very expensive for the server in terms of server CPU time and memory consumption. In fact, it has been shown that the evaluation problem for SPARQL is PSPACE-complete [63]. This makes hosting a public SPARQL endpoint a costly endeavour for publishers. Also, it contributes to SPARQL endpoint availability being magnitudes lower than that of regular HTTP servers [64], making them less useful for executing multiple tasks subsequently. However, in the case of *private* SPARQL endpoints, the number of users, requests, and the cost per request are under stricter control, and cache reuse is possibly higher, allowing for more efficient provisioning. Therefore, highly available endpoints in enterprise contexts can be viable and cost-effective. For instance, both the BBC's 2010 World Cup website [65] and Springer Nature's scolary data platform SciGraph[8] run on a SPARQL backend. Chapter 3 reprises the issues with public endpoints in more detail.

2.5.3 Linked Data documents

A more granular way to access Linked Data is publishing them according to the four Linked Data principles by Berners-Lee [17]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (e.g., RDF).
4. Include links to other URIs so that they can discover more things.

These principles divide an RDF dataset in several *Linked Data documents*, each of which contains triples related to a specific entity. Examples can be found with publishers such as DBpedia (http://dbpedia.org/page/DBpedia), the movie database LinkedMDB (http://data.linkedmdb.org/page/film/2014), or the multilingual encyclopedic dictionary BabelNet (http://babelnet.org/rdf/page). Although not standardized, there exists a de-facto definition formalized by Hartig [66] that is used in this dissertation:

Definition 1 (Linked Data document). A *Linked Data document* d_u is a collection of RDF triples that describes the entity identified by a URI u if there exists a triple $(s, p, o) \in d_u$ such that $s = u$ or $o = u$.

8. http://www.springernature.com/gp/researchers/scigraph

Note that there might be multiple Linked Data documents that describe an entity identified by u. However, according to the Linked Data principles [17], the URI u may also serve as a reference to a specific Linked Data document which is considered an authoritative source of data about the entity identified by u. The retrieval process of this document is called *dereferencing* the URI u. For instance, the URI http://dbpedia.org/resource/Umberto_Eco denotes the author Umberto Eco, and looking up this URI leads to a document with triples in which the URI is the subject or the object. This is a crucial property of Linked Data: if a client does not know what entity a URI represents, it can find information by performing an HTTP GET request.

From a REST perspective, the entity is the Web resource, the Linked Data document is its representation (e.g., in an RDF format or HTML), and all URIs are its hypermedia links. Hence, the REST notion of a "resource" thus coincides with the RDF notion of a "resource". For example, data about the RDF resource http://dbpedia.org/resource/Umberto_Eco is available through the REST resource with that same URL.

The server performance for responses can be high, and their cost is low, since the required data lookups to generate each document are light. Furthermore, the set of resources per dataset is finite, and resources can be reused by many clients, leading to high cache reuse. These factors allow high availability of the interface at low cost.

Dereferencing allows discovering related data from *one* particular authoritative source only. For instance, the aforementioned URL allows to discover information about Umberto Eco from the DBpedia [67] dataset, but not from any other sources with information about the same concept. Furthermore, discovery of one resource comes down to merely retrieving its entire representation, and does not directly give access to all resources in the dataset.

Hartig and Pirrò [68] studied the scope for SPARQL1.1 *Property Path* patterns i.e., a feature to express navigational routes [31], and introduced a classification according to their ability to be evaluated completely under context-based semantics, i.e., *Web-safeness*).

Several approaches exist to execute queries over Linked Data documents, as surveyed by Hartig [69]. One family of approaches uses pre-populated index structures [70], i.e., prefetching and preprocessing documents, and another focuses on live exploration by a traversal-based query execution [71], i.e., applying a follow-your-nose method by following links.

Typically, such query approaches have longer query execution times than SPARQL endpoints, but—unlike data dumps—allow for *live* querying. The required bandwidth is generally smaller than that of data dumps, but the efficiency can still be low depending on the type of query. For instance, chains of triple patterns connected with a variable (e.g., <s> `foaf:made ?o. ?o foaf:name` <o>.) can result in an explosion of HTTP requests. Furthermore, completeness with regard to a dataset cannot be guaranteed, and certain queries are difficult or impossible to evaluate without an index [69]. In particular, queries for patterns with unbound subjects (e.g., `?s foaf:made <o>`) pose problems, since Linked Data documents are by definition single-resource-centric—a client can only navigate using a resource's incoming and outgoing links.

2.5.4 Other specialized Web APIs to Linked Data

In addition to the (de facto) standard interfaces, several other HTTP interfaces for RDF exist. One important motivation was having *write* support. An early attempt for a read/write interface was the SPARQL Graph Store Protocol [72], part of the W3C SPARQL specification. It describes HTTP operations to manipulate RDF graphs through SPARQL queries, but is barely used. A later W3C recommendation, namely Linked Data Platform [73], defines a read/write Linked Data HTTP interface that adheres more to REST. The API details several concepts that extend beyond the Linked Data principles, such as containers and write access.

Linked Data Platform has already been implemented by several big platforms such as Apache Marmotta[9], OpenLink Vituoso[10], and TopBraid Live[11]. However, it was designed primarily for consistent read/write access to Linked Data resources, not to enable reliable and/or efficient query execution.

Additionally, there exist a few custom HTTP interfaces for triples, such as the Linked Data API [74] and Restpark [75]. Some of them aim to bridge the gap between the SPARQL protocol and the REST architectural style underlying the Web [76]. However, none of these proposals are widely used and no query engines for them are implemented to date.

2.6 Conclusion

The Web of Linked Data can be considered a practical embodiment of the Semantic Web. Therefore, this chapter first introduced the Web's core architectural aspects that are key for its success and allowed it to scale globally. Many of these aspects are incorporated in Semantic Web technologies to make Web content more understandable for machines. Supported by this technology stack, the Linked Open Data initiative aims to publish more machine-understandable datasets on the Web in an interconnected way. By querying these, applications can derive knowledge. To publish Linked Data on the Web, several Web interfaces exist. They all have their merits, but also show fallacies in enabling complex queries.

9. http://marmotta.apache.org
10. https://virtuoso.openlinksw.com
11. https://www.topquadrant.com/products/topbraid-live

This chapter was partly based on the publications:

Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. "Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web." In: *Journal of Web Semantics* 37–38 (Mar. 2016), pp. 184–206. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.03.003. URL: http://linkeddatafragments.org/publications/jws2016.pdf

Miel Vander Sande, Ruben Verborgh, Anastasia Dimou, Pieter Colpaert, and Erik Mannens. "Hypermedia-based discovery for source selection using low-cost Linked Data interfaces." In: *International Journal on Semantic Web and Information Systems* 12.3 (2016), pp. 79–110. ISSN: 1552-6283. DOI: 10.4018/ijswis.2016070103

References

[1] Sareh Aghaei, Mohammad Ali Nematbakhsh, and Hadi Khosravi Farsani. "Evolution of the World Wide Web: From WEB 1.0 TO WEB 4.0." In: *International Journal of Web & Semantic Technology* 3.1 (2012), p. 1.

[2] Ted H. Nelson. "Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate." In: *The 1965 20th National Conference.* ACM '65. Cleveland, Ohio, USA: ACM, 1965, pp. 84–100. DOI: 10.1145/800197.806036.

[3] Ray McAleese and Catherine Green. *Hypertext: state of the art.* Intellect Books, 1990.

[4] Maurice de Kunder. *The size of the World Wide Web (The Internet).* Aug. 31, 2017. URL: http://web.archive.org/web/20170831172922/http://www.worldwidewebsize.com.

[5] Antal Van den Bosch, Toine Bogers, and Maurice De Kunder. "Estimating search engine index size variability: a 9-year longitudinal study." In: *Scientometrics* 107.2 (2016), pp. 839–856.

[6] Chris Bizer, Tom Heath, Danny Ayers, and Yves Raimond. "Interlinking Open Data on the Web." In: *Demonstrations track, 4th European Semantic Web Conference, Innsbruck, Austria.* 2007.

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked Data – The Story So Far." In: *International Journal on Semantic Web and Information Systems* 5.3 (Mar. 2009), pp. 1–22.

[8] Tom Heath and Christian Bizer. "Linked data: Evolving the web into a global data space." In: *Synthesis lectures on the Semantic Web: theory and technology* 1.1 (2011), pp. 1–136.

[9] Tim Berners-Lee, Robert Cailliau, and Jean-François Groff. "The world-wide web." In: *Computer networks and ISDN systems* 25.4-5 (1992), pp. 454–459.

[10] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures." PhD thesis. University of California, 2000.

[11] Roy Thomas Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol (HTTP)*. Request For Comments 2616. Internet Engineering Task Force, June 1999. URL: http://tools.ietf.org/html/rfc2616.

[12] Ruben Verborgh. "Serendipitous Web Applications through Semantic Hypermedia." PhD thesis. Ghent, Belgium: Ghent University, Feb. 2014. URL: https://ruben.verborgh.org/phd/ruben-verborgh-phd.pdf.

[13] Mike Amundsen. "Hypermedia Types." In: *REST: From Research to Practice*. Ed. by Erik Wilde and Cesare Pautasso. Springer, 2011, pp. 93–116.

[14] Roy Thomas Fielding. *REST APIs must be hypertext-driven*. Oct. 2008. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[15] Tim Berners-Lee. "Sir Tim Berners-Lee: Raw data, now." In: *Wired UK* 2 (2012), p. 2013.

[16] Noor Huijboom and Tijs Van den Broek. "Open data: an international comparison of strategies." In: *European journal of ePractice* 12.1 (2011), pp. 4–16.

[17] Tim Berners-Lee. *Linked Data – Design issues*. Ed. by Tim Berners-Lee. July 27, 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html (visited on 06/18/2009).

[18] Anja Jentzsch, Richard Cyganiak, and Chris Bizer. *State of the LOD Cloud*. 2011. URL: http://lod-cloud.net/state/state%5C_2011.

[19] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. "Adoption of the Linked Data Best Practices in Different Topical Domains." English. In: *International Semantic Web Conference*. 2014, pp. 245–260. ISBN: 978-3-319-11963-2. DOI: 10.1007/978-3-319-11964-9_16.

[20] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/rdf11-concepts/.

[21] Michael Hausenblas. "Exploiting linked data to build web applications." In: *IEEE Internet Computing* 13.4 (2009), p. 68.

[22] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web." In: *Scientific American* 284.5 (May 2001), pp. 34–43.

[23] Tim Berners-Lee, Mark Fischetti, and Michael L Foreword By-Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation, 2000.

[24] Stan Franklin and Art Graesser. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents." In: *Intelligent agents III agent theories, architectures, and languages* (1997), pp. 21–35.

[25] James Hendler. "Is there an intelligent agent in your future?" In: *Nature* 11 (1999).

[26] Ronald J Brachman, Hector J Levesque, and Raymond Reiter. *Knowledge representation.* MIT press, 1992.

[27] Franz Baader. *The description logic handbook: Theory, implementation and applications.* Cambridge university press, 2003.

[28] Denny Vrandečić and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase." In: *Communications of the ACM* 57.10 (2014), pp. 78–85.

[29] Raymond Reiter. "On closed world data bases." In: *Logic and data bases.* Springer, 1978, pp. 55–76.

[30] Peter F Patel-Schneider and Ian Horrocks. "A comparison of two modelling paradigms in the Semantic Web." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.4 (2007), pp. 240–250.

[31] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language.* Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-query/.

[32] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. *SPARQL 1.1 Protocol.* Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-protocol/.

[33] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview.* Recommendation. World Wide Web Consortium, 2012. URL: https://www.w3.org/TR/owl2-overview/.

[34] Eric Miller. "An introduction to the Resource Description Framework." In: *Bulletin of the Association for Information Science and Technology* 25.1 (1998), pp. 15–19.

[35] World Wide Web Consortium et al. *World Wide Web Consortium publishes Public Draft of Resource Description Framework (RDF).* 1997. URL: http://www.w3.org/Press/RDF (visited on 09/16/2003).

[36] Fabien Gandon and Guus Schreiber. *RDF 1.1 XML syntax.* Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: https://www.w3.org/TR/rdf-syntax-grammar.

[37] David Beckett. *RDF 1.1 N-Triples.* Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/n-triples/.

[38] David Beckett, Tim Berners-Lee, Eric Prud́hommeaux, and Gavin Carothers. *RDF 1.1 Turtle.* Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/turtle/.

[39] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. *JSON-LD 1.0.* Recommendation. World Wide Web Consortium, Jan. 16, 2014. URL: http://www.w3.org/TR/json-ld/.

[40] Ashok Malhotra and Paul Biron. *XML Schema Part 2: Datatypes Second Edition.* Recommendation. World Wide Web Consortium, Oct. 28, 2004. URL: https://www.w3.org/TR/xmlschema-2/.

[41] Dan Brickley and Ramanathan V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema.* Recommendation. World Wide Web Consortium, Feb. 10, 2004. URL: http://www.w3.org/TR/rdf-schema/.

[42] Ceri Binding, Michael Charno, Stuart Jeffrey, Keith May, and Douglas Tudhope. "Template Based Semantic Integration:" English. In: *International Journal on Semantic Web and Information Systems* 11.1 (Jan. 2015), pp. 1–29. ISSN: 1552-6283. URL: http://www.igi-global.com/article/template-based-semantic-integration/135560.

[43] Panos Vassiliadis. "A survey of Extract–transform–Load technology." In: *International Journal of Data Warehousing and Mining (IJDWM)* 5.3 (2009), pp. 1–27.

[44] Kostas Patroumpas, Michalis Alexakis, Giorgos Giannopoulos, and Spiros Athanasiou. "TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples." In: *EDBT/ICDT Workshops.* 2014, pp. 275–278.

[45] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. "Triplify: light-weight linked data publication from relational databases." In: *The 18th international conference on World wide web.* ACM, 2009, pp. 621–630.

[46] James Clark et al. *XSL transformations (XSLT).* Recommendation. World Wide Web Consortium, 1999, p. 103. URL: http://www.w3.org/TR/xslt.

[47] Robert Signore, Michael O. Stegman, and John Creamer. *The ODBC solution: Open database connectivity in distributed environments.* McGraw-Hill, Inc., 1995.

[48] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. "Ontology-based data access: Ontop of databases." In: *International Semantic Web Conference.* Springer. 2013, pp. 558–573.

[49] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. "Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph." In: *The 23rd international conference on World wide web.* ACM. 2014, pp. 479–490.

[50] Juan F Sequeda and Daniel P Miranker. "Ultrawrap: SPARQL execution on relational data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 22 (2013), pp. 19–39.

[51] Srividya K. Bansal. "Towards a semantic extract-transform-load (ETL) framework for big data integration." In: *Big Data (BigData Congress), 2014 IEEE International Congress on.* IEEE, 2014, pp. 522–529.

[52] Satya S Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. *A survey of current approaches for mapping of relational databases to RDF.* Deliverable. W3CRDB2RDF Incubator Group, Jan. 8, 2009, pp. 113–130.

[53] Tomáš Knap, Maria Kukhar, Bohuslav Macháč, Petr Škoda, Jiří Tomeš, and Ján Vojt. "UnifiedViews: an ETL framework for sustainable RDF data processing." In: *European Semantic Web Conference*. Springer, 2014, pp. 379–383.

[54] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF mapping language*. Recommendation. World Wide Web Consortium, Sept. 27, 2012. URL: http://www.w3.org/TR/r2rml.

[55] Matthias Hert, Gerald Reif, and Harald C Gall. "A comparison of RDB-to-RDF mapping languages." In: *The 7th International Conference on Semantic Systems*. ACM, 2011, pp. 25–32.

[56] Anastasia Dimou, Miel Vander Sande, Jason Slepicka, Pedro Szekely, Erik Mannens, Craig Knoblock, and Rik Van de Walle. "Mapping hierarchical sources into RDF using the RML mapping language." In: *IEEE International Conference on Semantic Computing (ICSC)*. IEEE, 2014, pp. 151–158. DOI: 10.1109/ICSC.2014.25.

[57] Orri Erling and Ivan Mikhailov. "Virtuoso: RDF Support in a Native RDBMS." In: *Semantic Web Information Management*. Ed. by Roberto de Virgilio, Fausto Giunchiglia, and Letizia Tanca. Springer, 2010, pp. 501–519. ISBN: 978-3-642-04328-4.

[58] J Aasman. "AllegroGraph 4.0–industry's first real time RDF store." In: *Presentation at Semantic Technologies Conference (SemTech 2009), San Jose.* 2009.

[59] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. "Binary RDF Representation for Publication and Exchange (HDT)." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (Mar. 2013), pp. 22–41.

[60] Miguel A Martínez-Prieto, Carlos E Cuesta, Mario Arias, and Javier D Fernández. "The solid architecture for real-time management of big semantic data." In: *Future Generation Computer Systems* 47 (2015), pp. 62–79.

[61] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. "LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data." In: *The 13th International Semantic Web Conference*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, Oct. 2014, pp. 213–228.

[62] Michael Grobe. "RDF, Jena, SPARQL and the Semantic Web." In: *The 37th Annual ACM SIGUCCS Fall Conference: Communication and Collaboration*. 2009. ISBN: 978-1-60558-477-5.

[63] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL." In: *ACM Transactions on Database Systems* 34.3 (Sept. 2009), 16:1–16:45. ISSN: 0362-5915.

[64] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. "SPARQLWeb-Querying Infrastructure: Ready for Action?" In: *The 12th International Semantic Web Conference.* Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz. Nov. 2013.

[65] Jem Rayfield. *BBC World Cup 2010 dynamic semantic publishing.* May 1, 2010. URL: http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html.

[66] Olaf Hartig. "SPARQL for a Web of Linked Data: Semantics and Computability." In: *The Semantic Web: Research and Applications.* Ed. by Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti. Springer, 2012, pp. 8–23.

[67] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. "DBpedia—A crystallization point for the Web of Data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165. URL: http://www.websemanticsjournal.org/index.php/ps/article/view/164.

[68] Olaf Hartig and Giuseppe Pirrò. "A context-based semantics for SPARQL property paths over the web." In: *European Semantic Web Conference.* Springer. 2015, pp. 71–87.

[69] Olaf Hartig. "An Overview on Execution Strategies for Linked Data Queries." In: *Datenbank-Spektrum* 13.2 (2013), pp. 89–99.

[70] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. "Comparing Data Summaries for Processing Live Queries over Linked Data." In: *World Wide Web* 14.5–6 (2011), pp. 495–544.

[71] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. "Executing SPARQL Queries over the Web of Linked Data." In: *The 8th International Semantic Web Conference.* Ed. by Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan. Springer, 2009, pp. 293–309.

[72] Chimezie Ogbuji. *SPARQL 1.1 Graph Store HTTP Protocol.* Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-http-rdf-update/.

[73] Steve Speicher, John Arwe, and Ashok Malhotra. *Linked Data Platform 1.0.* Recommendation. World Wide Web Consortium, Feb. 26, 2015. URL: http://www.w3.org/TR/ldp/.

[74] Dave Reynolds. *Linked Data API.* Ed. by Dave Reynolds. 2015. URL: https://code.google.com/p/linked-data-api (visited on 08/02/2015).

[75] Luca Matteis. *Restpark: Minimal RESTful API for Retrieving RDF Triples.* 2013. URL: http://lmatteis.github.io/restpark/restpark.pdf.

[76] Erik Wilde and Michael Hausenblas. “RESTful SPARQL? You Name It! – Aligning SPARQL with REST and Resource Orientation.” In: *The 4th Workshop on Emerging Web Services Technology*. ACM, 2009, pp. 39–43. ISBN: 978-1-60558-776-9.

I'd trade it all for a little more.

— Mr. Burns

Sustainable APIs for Linked Data publishing 3

No one size fits all, and in publishing Linked Data on the Web, this is no different. However, the existing Linked Data interfaces seem to suggest otherwise. In terms of query execution, they cover two extremes in query expressiveness, forcing most of the load on either the publisher or the consumer. This chapter therefore introduces the Linked Data Fragments conceptual model to open up a wider spectrum of possible interfaces. First, we sketch how current Linked Data publishing practices restrict consuming applications, including the role of existing interfaces. Then, we introduce important characteristics to assess Web APIs, describe the Linked Data Fragments model, and explain how it can be applied to explore new interfaces.

Linked Data's main premise is that multiple datasets can be uniformly accessed and interpreted by both humans and machines, facilitating knowledge integration and sharing. The uniform interface enables answering more complex questions, possibly by combining multiple Linked Datasets. This potential has inspired organizations in various domains to adopt Linked Open Data principles and to start implementing them at scale [1].

One of these domains is the Libraries, Archives, and Museums (LAM) community, in which organizations acknowledged the benefits of Linked Data for their inherent focus on knowledge sharing [2]. A recent survey [3] on Linked Data adoption, indicates that the 96 LAM participants identified 172 projects or services being implemented. Of the 76 projects that were sufficiently described in the survey, 67% published Linked Data. The size of some available datasets in that area already ranges between tens of millions and billions of triples [4]. Prominent examples include WorldCat.org (15 billion), Europeana (4 billion), The European Library (2 billion), Library of Congress (500 million), and the British National Library (100 million). Efforts are currently ongoing in a wide range of domains [5], including electronic thesis and dissertations

(ETD) [6], image collections (e.g., the Getty museum[1]), and digital humanities (e.g., DARIAH[2], and cultural heritage (e.g., Pleiades[3]).

Another prominent domain is government. Here, the adoption of Linked Data originates from a policy enhancement perspective, perceived as desirable follow-up to Open Data (see Section 2.2). Shadbolt and O'Hara [7] state that, with Linked Data "inference will no longer be an oligopoly of governments and large corporations", which results in that "informed decision-making and service provisioning can be devolved to local governments, communities, the private sector, charities, nongovernmental organizations, and even individuals". Several government data portals already publish a significant amount of RDF, such as data.gov [8], data.gov.uk [9], and the EU Open Data Portal[4].

Finally, other large Linked Data producers are the Life Sciences domain [10] or community efforts such as DBpedia [11] and Wikidata [12].

These and similar efforts have lead to more Linked Datasets than ever before. The LODstats project [13] indicated that, as of May 2016, there are over 150 billion Linked Data triples distributed over 9,960 datasets. However, there seems to be a discrepancy between what exists and what can be queried on the public Web. Of the 9,960 datasets, only 2,973 are error-free (i.e., dumps cannot be parsed or downloaded; endpoints are down or fail to respond). Furthermore, 2,838 are dumps opposed to 151 SPARQL endpoints [14]. Despite that the majority of triples (98%) is served by the endpoints, this number is dominated by a minority of datasets. For *live* queryable data, downloadable RDF data dumps hardly qualify, as they imply high bandwidth usage and high client costs and weakened control for publishers; a queryable interface using SPARQL is preferred instead. Unfortunately, a recent RDF dump crawl collected over 38 billion triples[5], indicates that, along with the 900,129 crawled Linked Data documents [1], many small datasets out there fail to be published in a queryable way.

3.1 Towards sustainable Linked Data querying

In a Web of Linked Data, we cannot consider data dumps as the only access mechanism appropriate. After all, human consumers of Web content do not need to download an entire website before being able to read one or more webpages—so why should machine clients have to?

To facilitate complex querying in an interoperable way, support for SPARQL querying is recommended in a LOD environment [15]. Therefore, hosting a public *endpoint* is the most obvious choice [16]. Examples are Europeana [4] serving metadata on 2.4 million objects, the British Museum[6] publishing over

1. http://www.getty.edu/research/tools/vocabularies/lod
2. http://www.dariah.nl/
3. https://pleiades.stoa.org
4. http://data.europa.eu/euodp/en/linked-data
5. http://lodlaundromat.org
6. http://collection.britishmuseum.org/sparql

750,0000 records, and, of course, DBpedia offering access to over 1 billion RDF triples[7].

Sadly, these organizations are rare and the amount of live queryable Linked Datasets on the Web still appears to be low. With “live queryable”, we mean Linked Data that can be queried without first downloading the entire dataset. With “low availability”, we mean a two-sided problem:

- the majority of datasets is not published in queryable form [13]; and
- datasets that are published in a public SPARQL endpoint suffer from frequent downtime (i.e., more than 5% of the time on average) [17].

3.1.1 The caveats of public SPARQL endpoints

In March 2015, the LODstats project [13] listed 10,059 dataset descriptions, out of which only 464 referred to a SPARQL endpoint, and only 230 of these endpoints responded within 10 seconds with an HTTP `200 OK` status code to our `GET` request for their base URL. In 2016, only 151 error-free endpoints out of 9,960 datasets are listed, which is only 0.02% [14]. A 2013 survey revealed that the majority of public SPARQL endpoints had an uptime of less than 95% [17]. In other words, the average endpoint is down for more than 1.5 days each month. According to measurements conducted in 2016, the number of endpoints has tripled since 2011, but the availability rate has not improved; only 44.67% of 535 known endpoints have sufficient availability (> 95%) [18].

These facts make developing and running live applications on top of public endpoints an unrealistic endeavor in practice. This unavailability becomes all the more problematic if we consider queries over multiple, distributed datasets. As identified by a developer of the popular Virtuoso triple store, a practical use case for public SPARQL endpoints is to provide such endpoints “for lookups and discovery; sort of a dataset demo” [19]. If reliable access is needed, a common practice is to download a data dump and host a local endpoint. The scenario then becomes that of before, inheriting the same drawbacks.

> Some systems allow publishers to expose only a subset of all SPARQL queries; e.g., by placing restrictions on the allowed query execution times. However, even with such restrictions, the availability of public SPARQL endpoints remains low, not to mention the interoperability chaos it creates [17].

For an explanation, we reprise the architectural properties of the Web from Section 2.1. Exposing a query interface as expressive as SPARQL on the public Web contrasts with most other Web APIs, whose main purpose is to expose a *less* powerful interface than the underlying database. Doing the opposite evidently has a few important drawbacks.

In addition to the unpredictability of the total number of users and requests, the processing cost per request can vary significantly because of the relatively

7. https://dbpedia.org/sparql

high expressive power of SPARQL (with different fragments of SPARQL having a different computational complexity [20]). In most other Web APIs, the cost per request is bounded more strongly, because the interface is designed to restrict the query capabilities. Regardless of the absolute value of this cost, the high variations in the individual factors of the product make it difficult to correctly predict the necessary computational resources. Due to this unpredictability, infrastructure costs for a public SPARQL endpoint are generally high and the software systems complex. Furthermore, the relatively high expressive power causes clients to perform very individualized requests. As a consequence, caching such requests only allows for limited reuse, since other clients needing different requests is more likely.

3.1.2 Economical repercussions for publishers

While high availability is possible with a public SPARQL endpoint, it is potentially expensive. Not all Linked Data publishers enjoy sufficient financial resources and consider the maintenance cost of a reliable public SPARQL endpoint too high. But even the organizations that more or less do, like Europeana, British Museum, or DBpedia, are not able to combine a high number of users with an expected availability in "number of nines" [21, 22], counting the number of leading nines in the availability percentage. In the LAM community, for instance, institutions consider selecting infrastructure required to surface functional Linked Data (e.g. triplestores, SPARQL engines, indexing platforms) a high threshold for their projects [23]. Easily deployable, comprehensive publishing solutions are not available. In general, Linked Data *publishing costs* are higher than publishing those of well established traditional publishing solutions (e.g., Integrated Library Systems and Digital Asset Management systems) and as such present a proposition that is hardly feasible or sustainable [23]. Also, adoption of a Linked Data publication approach does not mean giving up on the traditional approach. Indeed, the traditional systems typically provide a wide variety of services beyond description and discovery. Hence, early Linked Data adopters have no other option than to invest in both. All the more reason infrastructure costs have to be as low as possible.

Rather than facing frequent unavailability, organizations prefer a more affordable, more reliable, but less expressive interface such as downloadable *data dumps* or Linked Data documents. Indirectly, SPARQL querying is still possible client-side, but is no longer live or fast. As described in Section 2.5, the complete dataset needs first to be downloaded, and, then, indexed in a local RDF database, or a lot of links need to be traversed first [24, 25]. Furthermore, their consumption is only possible on sufficiently powerful machines—not on mobile devices, whose popularity continues to increase—and requires a technical background to set up.

If we want Semantic Web applications on top of live Linked Datasets to become a reality, we must reconsider our options regarding Web-scale publication of Linked Data. Surely, current interfaces cannot cover the wide variety of possible use cases and their specific requirements. Between the two extremes of data dumps and SPARQL endpoints lies a whole spectrum of possible Web interfaces, which has remained largely unexplored. In particular, the practical aspects of Linked Data querying have been understudied so far [17]. Focus has been on query execution time, precision and recall, while the feasibility of most of SPARQL and Linked Data query approaches in a public Web setting is questionable. A Web context introduces many important characteristics, restrictions and opportunities, which are not mentioned or evaluated.

3.2 Preliminaries of the Web of Linked Data

Before we can have a closer inspection into interfaces, some general concepts on the Web of Linked Data need to be introduced first. The Web of Linked Data can be considered a collection of interconnected datasets, which are typically modeled in the RDF data model. Formally, the (infinite) set of all *RDF triples* can be considered $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, with as $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$, as the disjoint, infinite sets of URIs, blank nodes, and literals, respectively.

On the Web, these triples appear in digestable datasets, which contain entities, concepts and relationships about a specific domain in machine-readable semantics. We call such a dataset a *Linked Dataset.* Because blank nodes serve little purpose in a Web context—they are not identifyable by URI, hence they cannot serve as web resources (see Section 2.1.1)—and can be transformed into URIs via *skolemization* (i.e., by applying a template) [26], we assume for reasons of simplicity that a Linked Dataset does not contain blank nodes.

Definition 2 (Linked Dataset). A *Linked Dataset* $D \subseteq \mathcal{T}^*$ is a finite set of blank-node-free RDF triples, where $\mathcal{T}^* = \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$.

The HTTP URIs in $\mathcal{U}$ are native to the Web and, thus, directly accessible through HTTP methods. For instance, one can perform a simple HTTP request and expect a response. A successful response thereof is a Linked Data document discussed in Section 2.5.3. Retrieving a Linked Data document is thus a possible way to obtain a fraction of a Linked Dataset over HTTP. It is an example of a restricted *Linked Data interface* that only allows access to fragments about that particular URI. Other examples include data dumps, APIs, or SPARQL endpoints. These interfaces differ in their expressiveness, namely all data, simple queries, or complex queries.

Definition 3 (Linked Data interface). A *Linked Data interface* $i_D \in \mathcal{I}$ provides Web access to a Linked dataset D through interface-specific operations, with $\mathcal{I}$ as the set of all Web API interfaces.

3.3 Characterizing Web APIs

A Linked Data interface can be considered a Web Application Programming Interface (API) to RDF data. In Section 2.1.2, we introduced the Web API as an interface designed for automated, machine-based consumption of Web content. These come in two architectural styles: RPC and REST.

When used for publishing data—like in this context—a Web API generally has two main objectives:

- to abstract any database-specific aspects, such as schema or query language, from the client; and
- to restrict the type of queries clients can execute.

Interestingly, both are means for *scalability*, i.e., maximizing the number of data consumers that can be sustained over a long period of time. Databases are abstracted through the uniform HTTP interface, which scales the interoperability with clients. Restricting the types of queries protects the computational resources necessary to process them. Furthermore, it increases the chance that resources are reused through caching. Both balance the load on the server and, thus, scale the cost or infrastructure.

It is vital to acknowledge these objectives. In fact, negligence in this regard causes SPARQL endpoints to fail on the public Web (i.e., providing direct SPARQL access). Thus, what impacts the characteristics of the API, is its design in terms of *(a)* how the interface is defined, *(b)* what data it gives access to, and *(c)* with what query granularity.

In general, we distinguish four main API characteristics: *throughput*, *cost*, *cache reuse*, and *bandwidth*. These can apply to either servers in general or to clients performing a specific task [27]. The responses of an API are assumed to be complete and accurate. Table 3.1 describes the characteristics in more detail.

Given a task T a client needs to complete, two Web APIs I and I' might exhibit different behavior. For instance, a client might be able to complete T using a single large operation o with server-side cost c against I, whereas n smaller operations $o'_1, \dots, o'_n$ with costs $c'_1, \dots, c'_n$ might be needed in the case of I'. We assume here that the cost c'_i of each individual smaller operation o'_i is less than the cost of the large operation, c, but the total server cost for I' is $\sum_1^n c'_i$, which may be greater than c. If, however, some of these n smaller operations are cacheable, multiple clients executing tasks similar to T could reuse already generated responses from a cache, lowering the total number of requests—and thus the cost—for the server. Which of these factors dominates depends on the number of clients, the cost per request, the cache reuse ratio, the similarity of tasks, and other factors. In general, if costs increase to a certain level, a server might become fully occupied and unable to fulfill new incoming requests, and hence start a period of unavailability. It is thus important to examine how choosing a specific interface influences the cost for the server, to ensure high availability measurements.

Unsurprisingly, the choice in architectural style has an important impact on these characteristics. For instance, RPC-based APIs tend to involve more individualized and uncached requests, analogous to the one-on-one communication of software APIs and system APIs on local machines. REST APIs tend to

	Server	**Client**
Throughput *rate at which tasks can be completed*	The maximum number of requests it can handle per time interval (i.e., the inverse of the average request processing time).	The total number of API requests required to complete the task. Different APIs will require a different number of requests—depending on the request granularity an API offers—to fulfill tasks at the same rate.
Cost *amount of resources a request consumes*	The consumption of CPU, RAM, and I/O.	The time it takes to process one or more server responses into the desired result for a given task.
Cache reuse *ratio of items that are requested multiple times from the cache, versus the number of items it stores*	Lowering cost by subsequently serving responses, that are reusable by multiple clients, from a cache.	Priorly retrieved data that is reusable in the current task or future tasks.
Bandwidth *the total required size of HTTP communication*	The number of retrieved responses × the average response size (ignoring the relatively small request size). A portion of the responses might be cached, and thus involve cache bandwidth instead of server bandwidth.	The number of retrieved responses × the average response size (ignoring the relatively small request size).

Table 3.1: General characteristics of Web APIs

offer reusable resources that are consumed by multiple clients. For instance, a personalized webpage might still reuse images and other assets from a cache. Many individual variations are however possible, depending on the granularity of operations/resources in the API. Each variation burdens clients with restrictions that we need to take into account. Therefore, we introduce an important final characteristic to assess the practical use of the API, namely the *efficiency* for clients:

efficiency Efficiency for the client is the fraction of data retrieved from the server during the execution of a task over the amount of data required to execute that task.

3.4 The Linked Data Fragments conceptual model

As Web APIs always have thrived in diversity, the spectrum of Linked Data interfaces cannot be a story of two extremes. Interfaces offer either powerful query capabilities (e.g., SPARQL endpoints), or low server-side CPU cost (e.g., data dumps and Linked Data documents). With the task of query evaluation currently happening either fully on the server side, or fully on the client side, a sustainable Web of Linked Data that is queryable is not yet within reach.

Linked Data Fragments is a specification [28] maintained by the W3C Hydra community group.

To better balance these and other trade-offs in different ways, we designed the Linked Data Fragments (LDF) [29] conceptual model, a uniform view on all interfaces to RDF data. Thereby, it is easier to identify where the gaps are in an API's requirements, so they can be addressed. For instance, poor usability for the targeted applications that is caused by insufficient client performance, or low availability that is caused by excessive server cost. At the same time, it facilitates designing new interface trade-offs to hypothesize over, possibly leading to an advancement of the state-of-the-art in Linked Data publishing.

Throughout this section, I adopted the formal definitions for LDF performed by Hartig O. and Verborgh R. in [30] indicated by a gray backdrop like this one. These formalisms are lay the groundwork for some of the definitions and pseudo-code in Chapter 4, and are therefore indispensable.

3.4.1 Definition

Our model looks at what all Linked Data APIs have in common. In one way or another, they all publish a subset or *fragment* of a certain Linked Dataset. Hence, we uniformly call the response to each request a *Linked Data Fragment (LDF)*.

Each fragment returns a set of triples that were somehow selected, which is abstracted by the notion of a *selector*:

selector: the condition on which RDF triples are selected to be included in the LDF.

The selector largely shapes the URIs and thus how the Web API is partitioned in a set of possible HTTP resources. Therefore, it determines the specificity of the individual queries that clients can ask the server. An LDF can be fine-grained, like SPARQL (`CONSTRUCT`) queries, or course-grained, like a dereferenceable URI. Evidently, the selector directly impacts how the workload is balanced between client and server. Finding an optimal balance is important, as it increases the granularity of engagement Linked Data publishers can take [31]. This ability to optimize between API cost and efficiency, installs a lower threshold for publishing queryable Linked Data, ultimately leading to more available and easily consumable datasets.

Formally, the selector is considered a *function* on which triples can obtained from the Linked Dataset. For instance, it can select all triples of the dataset whose subject is the URI u or the triple-pattern-based selector functions defined later in Section 4.1.

Definition 4 (selector function). A *selector function* is a function that maps from $2^{\mathcal{T}^*}$ to $2^{\mathcal{T}^*}$.

In addition to the selector, three (possibly empty) parts compose an LDF, which we describe as follows:

data: RDF triples obtained from the dataset;
metadata: RDF triples that describe the data triples;
controls: links and forms to retrieve other related LDFs of the same or other datasets.

The *data* denotes the set of triples that were selected. The *metadata* can be any extra information about the fragment that can be useful to the client. This can be generic, like the name of the author, licensing information, or a description, or task-specific, like statistics for query optimization or filter operations.

The *controls* are hypermedia controls [32, 33] in accordance with the REST architectural style (discussed in Section 2.1.1), which LDF APIs can add to their representations. Formally, we consider those that do not alter resource state and, given some (possibly empty) input, cause a client to request a specific URL when activated.

Definition 5 (control). A (resource-state-invariant) *control* is a function that maps from some set to $\mathcal{U}$.

Controls can have zero arguments, i.e., a constant function such as URLs or hyperlinks, or multiple arguments, i.e., forms that can be invoked with field values. Particularly interesting are hypermedia controls that contain selectors and thus can generate URLs to new fragments. These allow clients to access more data from the same dataset.

Finally, given the concepts and definitions above, LDFs of a certain Linked Dataset D published by a Linked Data interface i_D, are formally defined as follows.

Definition 6 (Linked Data Fragment). Let $D \subseteq \mathcal{T}^*$ be a finite set of blank-node-free RDF triples. A *Linked Data Fragment (LDF)* of D is a tuple $f = \langle u, s, \Gamma, M, C \rangle$ with the following five elements:

- u is a URI (which is the "authoritative" source from which f can be retrieved);
- s is a selector function;
- Γ is a set of (blank-node-free) RDF triples that is the result of applying the selector function s to D, i.e., $\Gamma = s(D)$;
- M is a finite set of (additional) RDF triples, including triples that represent metadata for f;
- C is a finite set of controls.

By specifying the values for u, s, Γ, M, and C, any RDF-based data on the Web can be described as an LDF (we will apply this to define well-known Linked Data interfaces later on). However, note that this formalization (Definition 6) should be distiguished from its representation in a response to clients (see Section 4.1 and Section 4.4 for examples). For instance, even though a selector function is a component of their definitions, this function is not necessarily represented inside of the response.

3.4.2 Pages and collections

In some cases, an LDF's data can be quite large (i.e., the set Γ contains many elements), e.g., a data dump may contain millions of triples. To avoid forcing clients to download such a large fragment completely, a server that hosts LDFs may apply *paging*. If so, overly large responses are partitioned in to multiple pages. This enables clients to inspect the data only partially, or consume the metadata of a fragment without having to download its actual data. Formally, such a page is captured as follows.

Definition 7 (LDF page).
Let $f = \langle u, s, \Gamma, M, C \rangle$ be an LDF of some finite set of blank-node-free RDF triples $D \subseteq \mathcal{T}^*$. A *page partitioning* of f is a finite, nonempty set Φ whose elements are called *pages* of f and have the following properties:

1. Each page $\phi \in \Phi$ is a tuple $\phi = \langle u', u_f, s_f, \Gamma', M', C' \rangle$ with the following six elements: (i) u' is a URI from which page ϕ can be retrieved, where $u' \neq u$, (ii) $u_f = u$, (iii) $s_f = s$, (iv) $\Gamma' \subseteq \Gamma$, (v) $M' \supseteq M$, and (vi) $C' \supseteq C$.
2. For every pair of two distinct pages $\phi_i = \langle u'_i, u_f, s_f, \Gamma'_i, M'_i, C'_i \rangle \in \Phi$ and $\phi_j = \langle u'_j, u_f, s_f, \Gamma'_j, M'_j, C'_j \rangle \in \Phi$ it holds that $u'_i \neq u'_j$ and $\Gamma'_i \cap \Gamma'_j = \emptyset$.
3. $\Gamma = \bigcup_{\langle u', u_f, s_f, \Gamma', M', C' \rangle \in \Phi} \Gamma'$.
4. There exists a strict total order $\prec$ on Φ such that, for every pair of two pages $\phi_i = \langle u'_i, u_f, s_f, \Gamma'_i, M'_i, C'_i \rangle \in \Phi$ and $\phi_j = \langle u'_j, u_f, s_f, \Gamma'_j, M'_j, C'_j \rangle \in \Phi$ with ϕ_j being the direct successor of ϕ_i (i.e., $\phi_i \prec \phi_j$ and $\neg\exists \phi_k \in \Phi : \phi_i \prec \phi_k \prec \phi_j$), there exists a control $c \in C'_i$ with $u'_j \in \mathrm{img}(c)$.

Each page should contain *all* metadata and controls of the corresponding fragment, despite the presence of the control c for navigating from one page to the

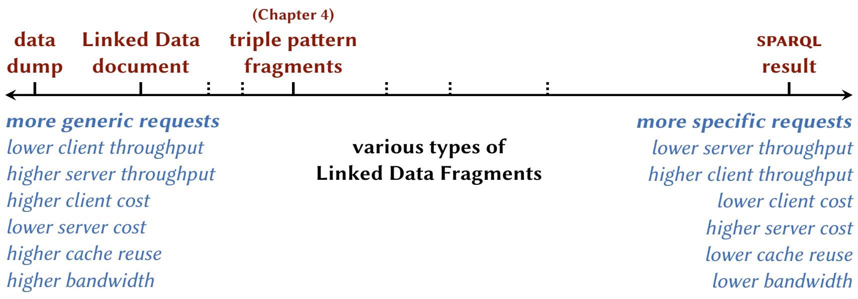


Figure 3.1: All Web APIs to RDF triples offer Linked Data Fragments of a dataset. These fragments differ in the specificity of the data they contain, and thus affect the cost to create them, the bandwidth to transfer them, their reusability through caching, and the overall performance of client and server.

next. If a server provides paging, it should avoid sending overly large chunks by redirecting clients from a fragment to the first page of the fragment. Also, similar to LDFs, the formalization of LDF pages (Definition 7) is independent from the representation in the response.

In many cases, a Web server that provides access to a Linked Dataset D exposes an interface i_D to retrieve multiple, different fragments of the dataset. As a last general concept, we define such a collection of LDFs.

Definition 8 (LDF collection)**.** Let $D \subseteq \mathcal{T}^*$ be a finite set of blank-node-free RDF triples, and let c be a hypermedia control. A c-specific *LDF collection* over D is a set F of LDFs such that, for each LDF $f \in F$ with $f = \langle u, s, \Gamma, M, C \rangle$, the following three properties hold:

1. f is an LDF of D;
2. $s \in \mathrm{dom}(c)$;
3. $c(s) = u$.

3.4.3 Application to existing interfaces

To put this framework into practice, we apply it to three existing interfaces: SPARQL endpoints, data dumps, and Linked Data documents. As demonstrated in Figure 3.1, they differ greatly in the granularity of the selector and, thus, the trade-off they impose according to the characteristics mentioned in Section 3.3. Table 3.2 summarizes these per interface.

A data dump as an LDF

A data dump of a dataset is a single fragment, hence, the selector is as follows:

selector: the identifier of the dump (e.g., the filename or URL).

		Throughput	**Cost**	**Cache reuse**	**Bandwidth**
Server	*Data dumps*	High	Low	Very high	High
	Linked Data documents	High	Low	Very high	Medium
	SPARQL endpoints	Low	High	Low	Low
Client	*Data dumps*	Low	High	High	High
	Linked Data documents	Low	Medium	High	High
	SPARQL endpoints	High	Low	Low	Low

Table 3.2: Data dumps, Linked Data documents, and SPARQL endpoints each impose a different trade-off mix of Web API characteristics

When requesting this identifier, the response is a fragment with the following components:

data: all triples of the dataset;
metadata: data about the dataset or the dump, such as version, author, or licensing details (the metadata set may be empty for some data dumps);
controls: empty, as the data dump contains the entire graph, so no controls to obtain other fragments are necessary.

RDF documents that contain HTTP URIs, contain hypermedia, including SPARQL result sets or RDF data dumps. Thus, we only explicitly mention them for Linked Data documents.

While some data dumps are divided across multiple archives (e.g., DBpedia 2014[8]), we can regard such archives as dumps of sub-graphs of a larger dataset. Hence, our conceptual framework still applies. However, their controls remain empty, as there is no hypermedia to navigate from one subgraph to another.

A SPARQL endpoint as LDFs

As our conceptual framework applies to interfaces that return triples from a dataset, we focus on only describing the `CONSTRUCT` query. Other clauses `SELECT` (returns solution mappings), `ASK` (returns a Boolean), `DESCRIBE` (is implementation-dependent [34]) cannot guarantee the correct subset of RDF triples. Note that creating a triple-based description of any other result form, such as solution mappings from `SELECT`, are possible. However, being conceptually similar, our focus on `CONSTRUCT` is no major limitation.

selector: defined based on the `CONSTRUCT` query.

8. http://wiki.dbpedia.org/Downloads2014

The response to a SPARQL `CONSTRUCT` query can be considered an LDF with the following properties:

data: all triples that are contained in the result of the query;
metadata: the metadata set is empty, as none are mentioned in the SPARQL specification;
controls: often, no explicit hypermedia controls are provided in the response because it is assumed that the client using the interface can extract the endpoint URL (e.g., `/sparql`) from the request URL (e.g., `/sparql?query=...`). This endpoint URL is also the control of the LDF collection exposed by a SPARQL endpoint.

As the use of `CONSTRUCT` queries entails, a selector may also return triples that are not contained in the dataset.

A Linked Data documents interface as LDFs

In accordance with Definition 1, a Linked Data document for a specific entity with URI u in a Linked Data documents interface has the following selector:

selector: implementation-specific which matches all triples that have u as subject or object.

Each document can therefore be considered an LDF with the following characteristics:

data: RDF triples that are related to the entity, e.g., having u as subject or object;
metadata may describe the provenance of the Linked Data document [35] or other types of metadata;
controls HTTP URIs in the data triples act as dereferenceable hyperlinks; both link to entities belonging to the same dataset as u are expected, as, in accordance with the Linked Data principles, links leading to other datasets.

Note that different fragments may have different hypermedia controls for this type of interface (which is not the case for data dumps or SPARQL endpoints).

3.5 Exploring new interface trade-offs

The Linked Data Fragments framework reveals a complete spectrum between Linked Data documents and the SPARQL protocol, in which the state-of-the-art of Linked Data publishing can be advanced. This enables exploration in the following three dimensions:

- **selector**, allowing more or less complex questions for the server;
- **controls**, modifying the response by adding or replacing navigational information for clients;
- **metadata**, modifying the response by adding or replacing any other information clients can use.

This doctoral work preserves focus by only studying variations in the *metadata* and *controls* dimension. As a selector, we use a triple pattern, which remains constant. The resulting Triple Pattern Fragments interface is introduced in the next chapter. In the remainder of this chapter, we first discuss the general approach for defining a new interface using *features* and how these can be evaluated.

3.5.1 Defining reusable interface features

Each variation in a certain dimension is materialized as a new *interface feature*, which is defined by Verborgh and Dumontier [36] as "a part of an interface that identifies, describes, and affords a certain kind of interaction across Web APIs". Common examples are full-text search, autocompletion, or file uploads. The goal is to move towards provider-independent, abstract interfaces, instead of the current practice of building many provider-specific interfaces.

Interface features enable reusable interface parts, both for the user and the API implementer. Clients can target one or more features instead of a specific API. Through the feature's interface, a client can determine how to be semantically compatible and how to invoke it. An API implementer can implement features as they see fit, but reuse a feature's interface, and thus possibly an existing implementation. The use of these features achieves a more extensive "loose-coupling" between client and server, but also enables better quantification to compare interfaces.

The experiments in this thesis test the introduced features in a practical context of *client-side SPARQL query execution* on *the public Web*. The focus on query execution ensures a client task, necessary to assess a Web API's performance, as mentioned in Section 3.3. The public Web context ensures the practical relevance and external validity of this work.

3.5.2 Evaluating interface features in public Web conditions

When evaluating new interface features in public Web conditions, three considerations exist. First, analogously to the Web itself, LDF interfaces should exist in a distributed, scalable manner in order to succeed. Generating additional metadata or controls introduces overhead on the server, which influences the ability to scale towards multiple clients. Second, the communication between client and server uses the HTTP protocol. Modeling, serialization, and compression determine the extra load of the overall network traffic. Third, with query execution on the client, novel approaches need to apply this metadata intelligently to increase efficiency.

Figure 3.2 illustrates a complete client-server setup and the five sequential levels impacted by interface features. To gain a clear picture for a particular feature, each level needs to be considered.

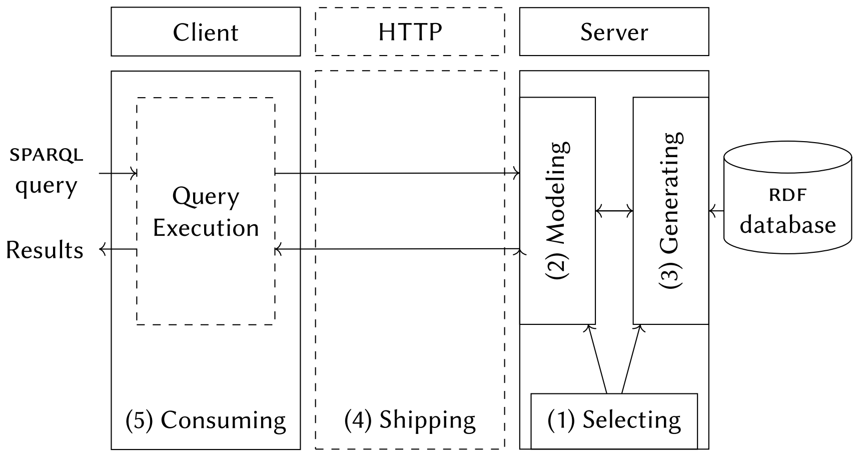


Figure 3.2: Complete setup with five sequential levels that are impacted by interface features and therefore subject for investigation.

selecting: a feature makes a good candidate if it is usable for both the context of RDF and the context of the Web, e.g., they are resistant to the delays and work well with the protocols and serializations.

generating: the necessary feature can be extracted or generated with different methods. We identify those who minimize server overhead, e.g., the average CPU usage, which directly impacts the cost of hosting such interface reliably.

modeling: we design *how* features are communicated to the client. To ensure the scalability of our approach in the distributed Web environment, we strive for *self-descriptive* interfaces. An RDF description on how the client should interpret the feature is included in the server's response, ensuring complete decoupling between client and server.

shipping: shipping (meta)data from server to client on the Web is subject to its characteristics: the HTTP protocol, the available network bandwidth, and the resource-oriented design. Therefore, we study techniques that improve the effects of different features on *caching* and *response size*. The former determines how a feature should be embedded in the request-response cycle. For example, considering a single request, is the metadata supplied in-band or as a separate resource? The latter dictates download speeds, thus requires optimal serialization or compression.

consuming: client-side query execution can be improved by using additional metadata or controls, provided by the interface. For instance, the number of required HTTP requests to solve a query can be lowered; or the recall of query results can be increased.

3.6 Conclusion

While Linked Data production may be on the rise in several domains, the availability of live queryable datasets on the Web remains low. The majority of datasets is not published in queryable form—they have to be downloaded first—and the ones that are published in a public SPARQL endpoint suffer from frequent downtime. Both issues are a consequence of the high costs to host a public SPARQL endpoint with high availability due to the language's expressiveness. Therefore, under-resourced publishers opt for inexpressive, but more affordable interfaces.

The Linked Data Fragments (LDF) [29] conceptual model allows us to analyze these existing Web interfaces to RDF in a uniform way. It helps determining the trade-off between client and server an interface offers in context of the Web API characteristics *throughput*, *cost*, *cache reuse*, and *bandwidth*. Based on that knowledge, we can identify the requirements, such as usability and availability, that should be addressed.

LDF can be used to explore new interfaces offering different combinations of characteristics. In the end, such advancements can lead to more sustainable live queryable Linked Data. Of crucial importance is the use of *interface features* that are reusable and comparable across APIs. They establish a loose coupling between client and server indispensable for scale and dealing with interface heterogeneity.

This chapter was partly based on the publications:

Miel Vander Sande. “Studying Metadata for better client-server trade-offs in Linked Data publishing.” In: *The Doctoral Consortium at the 15th International Semantic Web Conference (ISWC 2016).* Vol. 1733. CEUR, 2016, pp. 471–487

Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. “Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web.” In: *Journal of Web Semantics* 37–38 (Mar. 2016), pp. 184–206. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.03.003. URL: http://linkeddatafragments.org/publications/jws2016.pdf

Miel Vander Sande, Ruben Verborgh, Patrick Hochstenbach, and Herbert Van de Sompel. “Towards sustainable publishing and querying of distributed Linked Data archives.” In: *Journal of Documentation* (2017)

References

[1] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. “Adoption of the Linked Data Best Practices in Different Topical Domains.” English. In: *International Semantic Web Conference.* 2014, pp. 245–260. ISBN: 978-3-319-11963-2. DOI: 10.1007/978-3-319-11964-9_16.

[2] Seth van Hooland and Ruben Verborgh. *Linked Data for Libraries, Archives and Museums.* Facet Publishing, June 2014. ISBN: 978-1856049641.

[3] Karen Smith-Yoshimura. *Linked Data Survey results 1 – Who’s doing it.* 2014. URL: http://hangingtogether.org/?p=4137 (visited on 09/04/2014).

[4] Antoine Isaac and Bernhard Haslhofer. “Europeana Linked Open Data – data.europeana.eu.” In: *Semantic Web* 4.3 (2013), pp. 291–297.

[5] Erik T Mitchell. “The Current State of Linked Data in Libraries, Archives, and Museums.” In: *Library Technology Reports* 52.1 (2015), pp. 5–13.

[6] Lucas Mak, Devin Higgins, Aaron Collie, and Shawn Nicholson. “Enabling and integrating ETD repositories through linked data.” In: *Library Management* 35.4/5 (2014), pp. 284–292. DOI: 10.1108/LM-08-2013-0075. eprint: http://dx.doi.org/10.1108/LM-08-2013-0075.

[7] Nigel Shadbolt and Kieron O’Hara. “Linked data in government.” In: *IEEE Internet Computing* 17.4 (2013), pp. 72–77.

[8] James Hendler, Jeanne Holm, Chris Musialek, and George Thomas. “US government Linked Open Data: semantic.data.gov.” In: *IEEE Intelligent Systems* 27.3 (2012), pp. 25–31.

[9] Nigel Shadbolt, Kieron OH́ara, Tim Berners-Lee, Nicholas Gibbins, Hugh Glaser, Wendy Hall, et al. “Linked open government data: Lessons from data.gov.uk.” In: *IEEE Intelligent Systems* 27.3 (2012), pp. 16–24.

[10] Li Ding, Timothy Lebo, John S Erickson, Dominic DiFranzo, Gregory Todd Williams, Xian Li, James Michaelis, Alvaro Graves, Jin Guang Zheng, Zhenning Shangguan, et al. "TWC LOGD: A portal for linked open government data ecosystems." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 9.3 (2011), pp. 325–333.

[11] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. "DBpedia—A crystallization point for the Web of Data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165. URL: http://www.websemanticsjournal.org/index.php/ps/article/view/164.

[12] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. "Introducing Wikidata to the linked data web." In: *International Semantic Web Conference*. Springer, 2014, pp. 50–65.

[13] Ivan Ermilov, Michael Martin, Jens Lehmann, and Sören Auer. "Linked Open Data Statistics: Collection and Exploitation." In: *Knowledge Engineering and the Semantic Web*. Ed. by Pavel Klinov and Dmitry Mouromtsev. Vol. 394. Communications in Computer and Information Science. Springer, 2013, pp. 242–249. ISBN: 978-3-642-41359-9. DOI: 10.1007/978-3-642-41360-5_19.

[14] *LODStats*. AKSW. 2016. URL: http://stats.lod2.eu/ (visited on 05/01/2016).

[15] Erik T Mitchell. *Library linked data: Research and adoption*. American Library Association, 2013.

[16] Julia Marden, Carolyn Li-Madeo, Noreen Whysel, and Jeffrey Edelstein. "Linked Open Data for Cultural Heritage: Evolution of an Information Technology." In: *The 31st ACM International Conference on Design of Communication*. SIGDOC '13. Greenville, North Carolina, USA: ACM, 2013, pp. 107–112. ISBN: 978-1-4503-2131-0. DOI: 10.1145/2507065.2507103.

[17] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. "SPARQLWeb-Querying Infrastructure: Ready for Action?" In: *The 12th International Semantic Web Conference*. Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz. Nov. 2013.

[18] Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil-Aranda. "SPARQLES: Monitoring public SPARQL endpoints." In: *Semantic Web* Preprint (2016), pp. 1–17.

[19] Orri Erling. *SEMANTiCS 2014 (part 3 of 3): Conversations*. Aug. 2014. URL: https://www.openlinksw.com/dataspace/doc/oerling/weblog/Orri%5C%20Erling%5C%27s%5C%20Blog/1815.

[20] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL." In: *ACM Transactions on Database Systems* 34.3 (Sept. 2009), 16:1–16:45. ISSN: 0362-5915.

[21] Manish Marwah et al. “Quantifying the Sustainability Impact of Data Center Availability.” In: *SIGMETRICS Performance Evaluation Review* 37.4 (Mar. 2010), pp. 64–68. ISSN: 0163-5999. DOI: 10.1145/1773394.1773405.

[22] James E. J. Bottomley. “Implementing Clusters for High Availability.” In: *The Annual Conference on USENIX Annual Technical Conference.* ATEC ’04. Boston, MA: USENIX Association, 2004, pp. 44–44. URL: http://dl.acm.org/citation.cfm?id=1247415.1247459.

[23] T Erik et al. “The Evolving Direction of LD Research and Practice.” In: *Library Technology Reports* 52.1 (2015), pp. 29–33.

[24] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. “Comparing Data Summaries for Processing Live Queries over Linked Data.” In: *World Wide Web* 14.5–6 (2011), pp. 495–544.

[25] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. “Executing SPARQL Queries over the Web of Linked Data.” In: *The 8th International Semantic Web Conference.* Ed. by Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan. Springer, 2009, pp. 293–309.

[26] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax.* Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/rdf11-concepts/.

[27] Ruben Verborgh, Erik Mannens, and Rik Van de Walle. “Bottom-up Web APIs with self-descriptive responses.” In: *The 6th International Workshop on Modeling Social Media.* Feb. 2015.

[28] Ruben Verborgh. *Linked Data Fragments.* Unofficial Draft. Hydra W3C Community Group, June 5, 2016. URL: http://www.hydra-cg.com/spec/latest/linked-data-fragments/.

[29] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. “Web-scale querying through linked data fragments.” In: *7th Workshop on Linked Data on the Web, Proceedings.* Seoul, Korea, 2014, p. 10.

[30] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. “Querying Datasets on the Web with High Availability.” In: *The 13th International Semantic Web Conference.* Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, Oct. 2014, pp. 180–196.

[31] Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. “Linked Data-as-a-Service: The Semantic Web Redeployed.” In: *European Semantic Web Conference.* Ed. by Fabien Gandon, Marta Sabou, Harald Sack, Claudia d’Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann. Springer International Publishing, June 2015, pp. 471–487.

[32] Mike Amundsen. “Hypermedia Types.” In: *REST: From Research to Practice.* Ed. by Erik Wilde and Cesare Pautasso. Springer, 2011, pp. 93–116.

[33] Roy Thomas Fielding. *REST APIs must be hypertext-driven.* Oct. 2008. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[34] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language.* Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-query/.

[35] Olaf Hartig and Jun Zhao. “Publishing and Consuming Provenance Metadata on the Web of Linked Data.” In: *The 3rd International Provenance and Annotation Workshop.* Ed. by Deborah L. McGuinness, James R. Michaelis, and Luc Moreau. June 2010.

[36] Ruben Verborgh and Michel Dumontier. “A Web API ecosystem through feature-based reuse.” In: *arXiv preprint arXiv:1609.07108* (2016).

When will I learn? The answer to life's problems aren't at the bottom of a bottle, they're on TV!

— Homer J. Simpson

Query Execution 4

If we want to make sense of Linked Data, we need more sustainable Linked Data interfaces to query it. In this case, sustainability desires solutions with minimal server complexity (minimizing the cost *for data publishers) while still enabling live querying (maximizing the* utility *for Linked Data applications). This chapter therefore addresses Research Question 1 and Research Question 2. To this end, we introduce the Triple Pattern Fragments interface, a novel trade-off interface which restricts queries to triple patterns and includes cardinality metadata. We demonstrate client-side query processing using this interface and investigate whether it can preserve a low server cost (Hypothesis 1). Accordingly, we evaluate adding approximate membership metadata to the interface in context of the public Web and its characteristics (Hypothesis 2).*

Intelligent applications consume a Web of Linked Data by querying it. Unfortunately, due to the availability issues of SPARQL endpoints [1] and a lack of efficient alternatives [2], this is not yet possible in a sustainable way. This inconvenient truth prevents Linked Data applications to offer a reliable service to users, which reveals an urgency for exploring interfaces that offer new trade-offs.

The Linked Data Fragments (LDF) framework not only enables us to analyze existing Web APIs to RDF in a uniform way, it also allows us to define new APIs with a different combination of characteristics. Hence, we envisioned an interface that facilitates SPARQL query execution, but deals with unavailability issues by *(a)* limiting the cost for the server, and *(b)* ensure a high reuse of cached responses. These are intentionally aspects of scalability retrieved from the Web's architecture and REST constraints discussed in Section 2.1.1. Both aspects can be achieved by using a resource partitioning more granular than Linked Data documents, but far less expressive than SPARQL endpoints. Accordingly, we developed *Triple Pattern Fragments* (TPF), a low-cost interface to RDF triples.

With a granularity coarser than a SPARQL endpoint, executing a SPARQL query over a TPF interface is still possible, but the query needs to be split client-side into several sequential requests. To optimize this process—and to reduce the number of requests needed per client—the interface also provides *metadata* about its RDF dataset.

Using statistical and other metadata to optimize the query process has been an extensive research topic in databases for decades [3], and has been applied to SPARQL over RDF data. However, most efforts so far assume a centralized or local database environment (e.g., RDF triple stores). In a Web environment, the request-response paradigm, public network delay (which should stay below the common HTTP request timeout of ±30s), and preserving the uniform interface, quickly restrict the applicability of many metadata types. For instance, a histogram [4] is an efficient metadata type for selectivity estimation, but is cumbersome to serialize and produces a sizable HTTP response.

This chapter evaluates *cardinality* metadata—inherent to TPF—and *approximate membership* metadata in a decentralized public Web setting. These metadata types were selected because they are easily calculated or estimated, easily shipped (because of their compactness), and easily semantically described. The choice for AMF also resulted from the query algorithm, as we will explain in Section 4.4.

Finally, to avoid the inflation of custom APIs for which dedicated clients are needed [5], we supply TPF with hypermedia controls. Such controls allow clients to identify whether they are talking to a TPF interface or some other interface, and how to access TPFs through it. A hypermedia-aware client should be able to access the API by using hypermedia controls provided in the responses [6].

4.1 Triple Pattern Fragments: a new interface trade-off

To properly position Triple Pattern Fragments among existing interfaces, we define it by applying the LDF model and discuss its response format.

Throughout this chapter, I adopted the formal definitions for TPF performed by Hartig O. and Verborgh R. in [7], which build on the formalisms from Section 3.4 and are indicated by this gray backdrop. Again, they are included because they are essential for understanding some of the definitions and pseudocode in Chapters 5 to 7.

4.1.1 Definition

As the name suggest, this interface provides *triple-pattern*-based access to RDF datasets. This basic element of SPARQL was briefly introduced in Section 2.3.2 and can be written formally in the following fashion.

Given that $\mathcal{V}$ is the infinite set of all variables that is disjoint from the sets $\mathcal{U}$ (the set of all URIs), $\mathcal{B}$ (all blank nodes), and $\mathcal{L}$ (all literals), a *triple pattern* is any tuple $tp \in (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V})$.

Proceeding with an LDF-compliant definition, the selector of the TPF interface is as follows.

selector: a triple pattern; a triple is returned when its subject, predicate, or object match the value or variable of the corresponding term.

This pinpoints an interesting compromise along the axis in Figure 3.1, because *(a)* triple patterns are the most basic building blocks of SPARQL queries [8], and *(b)* low-cost indexes exist that efficiently return triples that match a given triple pattern [9]. The former facilitates client-side querying, as each fragment directly answers a part of the query, while the latter helps maximizing server-side availability through minimal CPU and memory cost. Formally, TPF introduces a specific type of selector function.

Definition 9 (triple-pattern-based selector function). Let tp be a triple pattern. The *triple-pattern-based selector function* for tp, denoted by s_{tp}, is a selector function that, for every set $D \in 2^{\mathcal{T}^*}$, is defined by $\mathrm{s}_{tp}(D) = \{t \in D \mid t \text{ is a matching triple for } tp\}$.

A TPF interface provides access to Linked Data Fragments with the following properties:

data: all triples of a dataset that match a given triple pattern;

metadata: an estimate of the number of triples that match the given triple pattern, i.e., the *cardinality*;

controls: a hypermedia form that allows clients to retrieve any TPF of the same dataset.

The Restpark API [10] was designed around the same time as TPF. Coincidentally, they are very much alike—Restpark only defines the triple-pattern-based selector, but shares the same motivation: some innovation happens through simplification.

As mentioned in Section 3.4.2, a TPF server should divide each fragment into reasonably sized pages to prevent clients accidentally downloading very large chunks. For instance, a response for a triple pattern with three unbound variables would contain *all* triples of the dataset if it is not paged. This explains the presence of the cardinality metadata: like in any paged interface on the Web, it is beneficial to know how many items there are in total since each page only shows a part of the whole. The default page size of TPF was fixed at 100 data triples; its effect is discussed later on in Section 4.3.3.

Note that the TPF interface requires the presence of a cardinality estimation, but does not define how or where this estimation is done. Typically, this responsibility is left to the *datasource*—the database or index that stores the Linked Datasets to which the TPF interface provides access. This is not an unrealistic demand; cardinality estimation is a common feature in databases, because it can be done cheap and efficiently (exact counting is more difficult).

A good example thereof is HDT (see Section 2.4.2), which is why it is currently the recommended datasource for TPF interfaces.

A server could wield a variable page size, e.g., an approximated optimum based on run-time observations (possibly with machine learning). However, this impacts resource partitioning significantly, which limits caching. More interesting could be fixed page sizes per triple pattern based on its estimated selectivity—less selective means bigger pages.

The hypermedia controls or forms consist of three fields (`subject`, `predicate`, `object`), such that the server offers a finite number of resources that correspond to triple-pattern queries. For instance, the result of a query for (`?s`, `?p`, `foaf:Person`) on a given dataset could be available as the resource http://fragments.dbpedia.org/en?object=http%3A%2F%2Fxmlns.com%2Ffoaf%2F0.1%2FPerson. A hypermedia-aware client accesses the API by using hypermedia controls provided in the responses [6]. The structure is deliberately not defined in any specification, as we define an interface feature (Section 3.5), not the API itself.

We conclude with the combined—they are closely connected—formal definition of Triple Pattern Fragment and a Triple Pattern Fragment collection.

Definition 10 (Triple Pattern Fragment)**.** Given a control c, a c-specific LDF collection F is called a *Triple Pattern Fragment collection* if, for every possible triple pattern tp, there exists one LDF $\langle u, s, \Gamma, M, C\rangle \in F$, referred to as a *Triple Pattern Fragment*, that has the following properties:

1. s is the triple-pattern-based selector function for tp.
2. There exists a triple $\langle u, \texttt{void:triples}, cnt\rangle \in M$ with cnt representing an estimate of the cardinality of Γ, that is, cnt is an integer that has the following two properties:
   a) If $\Gamma = \emptyset$, then $cnt = 0$.
   b) If $\Gamma \neq \emptyset$, then $cnt > 0$ and $\text{abs}\big(|\Gamma| - cnt\big) \leq \epsilon$ for some F-specific threshold ϵ.
3. $c \in C$.

The threshold ϵ accommodates for implementation differences in calculating the approximate number of triples matching a given triple pattern; interfaces are not required to return the exact number, but they should strive to minimize ϵ. By Definition 10, TPFs have to include the collection-specific hypermedia control, which makes any TPF collection a hypermedia-driven REST API [6]. Hence, by discovering an arbitrary fragment of such a collection, TPF clients can directly reach and retrieve *all* fragments of the collection, which covers the complete set of all possible triple patterns.

4.1.2 Response format

The manner in which servers can represent TPF resources is defined in the TPF specification [11]. When a fragment is requested, the HTTP response should include the data, metadata, and controls, as the RDF serialization in Listing 4.1

```
 1 <http://fragments.dbpedia.org/2014/en#metadata> {
 2   <http://fragments.dbpedia.org/2014/en#metadata> foaf:primaryTopic <>.
 3   <http://fragments.dbpedia.org/#dataset>
 4     hydra:member <http://fragments.dbpedia.org/2014/en#dataset>.
 5
 6
 7   <http://fragments.dbpedia.org/2014/en#dataset> a void:Dataset, hydra:Collection;
 8    void:subset <http://fragments.dbpedia.org/2014/en>;
 9    # Controls
10    hydra:search [
11     hydra:variableRepresentation hydra:ExplicitRepresentation;
12     hydra:template "http://fragments.dbpedia.org/2014/en{?s,p,o}";
13     hydra:mapping [ hydra:variable "s"; hydra:property rdf:subject    ],
14                   [ hydra:variable "p"; hydra:property rdf:predicate. ],
15                   [ hydra:variable "o"; hydra:property rdf:object     ]
16    ].
17
18   <http://fragments.dbpedia.org/2014/en> a hydra:PartialCollectionView;
19    hydra:itemsPerPage "100"^^xsd:integer;
20    hydra:first <http://fragments.dbpedia.org/2014/en?page=1>;
21    hydra:next <http://fragments.dbpedia.org/2014/en?page=2>
22    # Metadata
23    dcterms:title "Linked Data Fragment of DBpedia 2014"@en;
24    dcterms:description "Triple Pattern Fragment of the 'DBpedia 2014'
25                         dataset containing triples matching the pattern { ?s ?p ?o
                               }."@en;
26    dcterms:source <http://fragments.dbpedia.org/2014/en#dataset>;
27    void:triples "377367913"^^xsd:integer;
28 }
29
30 # Data
31 dbpedia:Barack_Obama a foaf:Person;
32 ...
```

Listing 4.1: The metadata graph in TriG describes a three-field form and paging information using the Hydra Core Vocabulary.

illustrates. This is *self-descriptive*, meaning that *(a)* extensions to the TPF interface are features of a composable API, ensuring backward-compatibility; *(b)* clients can discover at run time which features are supported. Note that preference is given to the quad-based formats N-Quads [12], TriG [13], or JSON-LD [14] to properly separate the data triples from the others by using a named graph. However, triple-based formats can be considered if content negotiation reveals the client does not support quads.

Line 1 indicates the start of the RDF graph that contains the metadata, which describes the given document (line 2). Next, line 7 defines a dataset or TPF collection—in this case DBpedia 2014—that is a member of all datasets hosted on this domain (lines 3 and 4).

To stimulate community participation, the TPF specification is maintained by the Hydra W3C Community Group.

Lines 10 to 16 describe the hypermedia controls using the Hydra Core Vocabulary [15], which allows Web APIs to describe the RDF equivalent of HTML links and forms (as mentioned in Section 2.1). Since these controls are used to search the entire dataset, and not the fragment or page, they are attached to the *dataset* resource (line 7). A templated link [16] offers a similar functionality to HTML forms (line 12). To construct new TPF URLs, a client can fill in `rdf:subject`, `rdf:predicate`, and `rdf:object`, which are mapped to s, p, and o (lines 13 to 15). The choice of field names is not imposed, as the in-band hypermedia avoids the need to standardize these names and/or the TPF URL structure.

Line 27 describes the cardinality of the fragment as an `xsd:integer`, enclosed in a triple with predicate `void:triples`. The amount of data triples in the response will be a lot less, as a TPF should be paged; hence the added information on page size (line 19) and links for navigating from one page to another (lines 20 and 21). The data component is simply represented as RDF triples added to the default graph (line 31).

4.2 Querying a TPF interface with SPARQL

To use the TPF interface to execute SPARQL queries over remote Linked Datasets, we introduce a TPF query client.

4.2.1 Client architecture

Generally speaking, a client accepts a SPARQL query and the URI of an arbitrary page of some fragment of the TPF collection. As mentioned before, any such TPF page contains a hypermedia control informing the client that the interfaces supports lookups by triple patterns. After obtaining such a hypermedia control, the client executes the query by applying a query processing algorithm.

To anticipate on future features added to the TPF interface, like the ones that will be introduced in Chapters 5 to 7, we propose the three-tier architecture shown in Figure 4.1. There are three layers in total:

1. the **Query Engine**;
2. the **Hypermedia Layer**;
3. the **HTTP Layer**.

The Query Engine uses a *query decomposition* process to recursively isolate triple patterns from the remaining graph pattern, and a *query execution algorithm* that determines the order in which that occurs. A single SPARQL query therefore results in several requests to the server, which are ultimately performed by the HTTP layer. In between, the Hypermedia Layer is responsible for translating every request for a triple pattern into a concrete HTTP request for a TPF.

At the end of this section, we describe the query execution algorithm that is used in our experiments. As more interface features will be introduced in the next chapters, this algorithm remains constant throughout this dissertation. This ensures that the impact of these interface features can be measured accurately. The three-tier client model ensures no modifications to the algorithm

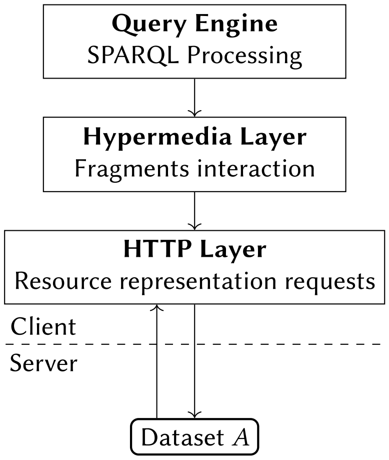


Figure 4.1: A three-tier architecture for the TPF client facilitates adding support for future interface features without many alterations.

or its implementation are necessary. Before we proceed to the explanation, however, we recollect some standard definitions of SPARQL to ensure a better understanding.

4.2.2 Preliminaries of SPARQL

As already introduced in Section 2.3.2, a SPARQL query is commonly centered around a *basic graph pattern* (BGP). Such BGP is embedded in the `WHERE` clause of the query, as shown in Listing 4.2 and defined by the SPARQL specification [8].

A BGP, usually denoted by B, is considered any finite set of triple patterns. Triple patterns (or BGPs) can be combined with specific operators [8, 17] (e.g., `FILTER` or `UNION`; see Section 2.3) in other SPARQL *graph patterns* P. For any such pattern P, the set of all variables that occur in P is denoted by $\text{vars}(P)$.

Next, we look at the standard (set-based) query semantics for SPARQL.

The *query result* of a graph pattern P over a Linked Dataset (of RDF triples) D is defined as a set denoted by $[\![P]\!]_D$ and that consists of so called *solution mappings*, that is, partial mappings $\mu : \mathcal{V} \longrightarrow (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$.

An RDF triple t is a *matching triple* for a triple pattern tp if there exists a solution mapping μ such that $t = \mu[tp]$, where $\mu[tp]$ denotes the triple (pattern) that we obtain by replacing the variables in tp according to μ. Usage of $\mu[tp]$ can be extended to a BGP B by applying μ to every triple pattern in B, which we denote by $\mu[B]$.

Finally, in context of the query algorithm that is about to be introduced, we also include the semantics of SPARQL graph patterns when used to query a Linked

Dataset that is published as a collection of TPFs. Specifically, the following definition specifies the expected query result of evaluating a SPARQL graph pattern over a TPF collection.

Definition 11 (query semantics). Let $D \subseteq \mathcal{T}^*$ be a finite set of blank-node-free RDF triples, and let F be some TPF collection over D. The *evaluation* of a SPARQL graph pattern P over F, denoted by[a] $[\![P]\!]_F$, is defined by $[\![P]\!]_F = [\![P]\!]_D$.

a. As usual when introducing multiple evaluation functions to define query semantics (e.g., [18]), we overload notation here. That is, depending on the object represented by the subscript, $[\![P]\!]_.$ denotes the evaluation of P over a set of RDF triples or over a TPF collection.

Observe that, by Definition 11, an approach to execute SPARQL queries over TPF collections is sound and complete if and only if the approach returns query results that are equivalent to the results expected from evaluating the queries directly over the Linked Datasets exposed as TPF collections. While this requirement seems trivial in the context of TPF collections, we emphasize that for other types of LDF collections it is not necessarily possible to ensure such an equivalence. For instance, query evaluation over a Linked Data documents interface (which is an LDF collection) cannot be guaranteed to be complete with respect to all data in the Linked Dataset that is exposed by the interface [19, 20] (unless all links between the Linked Data documents are bidirectional).

4.2.3 Query engine

An execution tree is often represented in a (relational) algebra, so it can optimized and simplified easily. Also SPARQL has such algebra described in its specification [8], but for the sake of simplicity, this is not discussed here.

In short, a complex SPARQL query is evaluated by first transforming the query into a query tree (query decomposition). The root node of that tree represents the query result, while a leaf node represents each triple pattern in the query. In between, each node in the tree represents an intermediate operation on the partial mappings μ. These partial mappings originate from the leaf nodes that each correspond to a Triple Pattern Fragment request. The majority of operations are extracted from the `WHERE` clause of the query; only operations that apply to the final resultset like `DISTINCT` are not.

The sequence of operation is directed from the leaves to the root, eventually leading to a complete result. One particular operation is the basic graph pattern B, which combines triple patterns conjunctively according to their shared variables. Thus, a B node will always be directly above multiple triple pattern leafs in the tree. Other possible operation nodes represent other SPARQL constructs that were mentioned in Section 2.3.2, such as `FILTER`, `UNION` or `DINSTINCT`. A SPARQL graph pattern P which contains a `FILTER` clause, therefore consists of two directly connected operation nodes: one bottom node for B, and one top node for `FILTER`.

```
SELECT ?person ?city WHERE {
    ?person a dbpedia-owl:Architect.                       # tp_1 (± 2,300 matches)
    ?person dbpprop:birthPlace ?city.                      # tp_2 (± 572,000 matches)
    ?city dc:subject dbpedia:Category:Capitals_in_Europe.  # tp_3 (± 60 matches)
} LIMIT 100
```

Listing 4.2: This SPARQL query finds architects born in European capitals.

For reasons of simplicity, we will only discuss a query execution algorithm limited to the evaluation of BGPs without special clauses. It determines how a B operation node, which combines several TPFs, processes partial mappings μ. Basically, the fragment cardinality metadata is used to determine an efficient execution order (query execution algorithm). In order to enable *incremental results*, this is implemented using the common iterator model [21] as basis. Iterators allow a consumer to obtain individual results of each operation separately, one at the time; hence, they are popular in query execution systems for computing results incrementally, and increasing the flexibility of implementing query operators [22].

Our algorithm executes queries by constructing a tree of iterators, each providing three operation functions:

- `Open` initializes the data structures needed to perform the operation,
- `GetNext` returns the next result of the operation, and
- `Close` ends the iteration and releases allocated resources.

Such an iterator tree computes a query result in a *pull*-based fashion; that is, during execution, the `GetNext` function of each iterator calls `GetNext` on the child(ren) and uses the input obtained by these calls to produce the next result(s). In other words, the algorithm constructs a pipeline of iterators where each iterator is the *source iterator* I_s of the next iterator.

In total, we employ three types of *non-blocking* iterators, i.e., results can be returned without having to pull in all input results first.

TriplePatternIterator: generates solution mappings for a triple pattern using a TPF interface;

BasicGraphPatternIterator: generates solution mappings for a basic graph pattern using a TPF interface and a combination of multiple `TriplePatternIterators`;

RootIterator: an initialization-free helper that returns an empty mapping $\mu_\emptyset$ exactly once, and `nil` after that. It is used as a starting source iterator.

A `TriplePatternIterator` (Listing 4.3) reads solution mappings from a source iterator I_s and combines them with possible mappings for a given triple pattern tp. For instance, assume the client is given the query in Listing 4.2, and the URI http://fragments.dbpedia.org/2015/en to retrieve a page from a TPF interface to the DBpedia 2015 dataset. If its triple pattern is ⟨`?person`, `dbpprop:birthPlace`, `?city`⟩ and the source iterator has returned

μ_s = {*?city* $\mapsto$ `dbpedia:Amsterdam`}, the `TriplePatternIterator` will subsequently request ⟨`?person`, `dbpprop:birthPlace`, `dbpedia:Amsterdam`⟩ triples through the TPF interface. If the TPF response contains a triple ⟨`dbpedia:Erick_van_Egeraat`, `dbpprop:birthPlace`, `dbpedia:Amsterdam`⟩, then the iterator will return the combined solution mapping $\mu \cup \mu_s$ = {*?city* $\mapsto$ `dbpedia:Amsterdam`, *?person* $\mapsto$ `dbpedia:Erick_van_Egeraat`}. A `TriplePatternIterator` has two member variables: `self.`ϕ to hold the current TPF page from which it is reading, and `self.`μ_s to hold the most recently read mapping from `self.`ϕ. If this page is read, the next page of the same TPF is requested if it exists (line 5). If the fragment has no more pages, the next mapping `self.`μ_s is read from the source iterator I_s, and the first page of the TPF for the mapped triple pattern `self.`$\mu_s[tp]$ is fetched (lines 7 to 9). That way, the solution mappings resulting from matching triples in the TPF's pages are compatible with the corresponding input mappings `self.`μ_s.

```
1  Function TriplePatternIterator.GetNext()
       Data: A source iterator $I_s$; a triple pattern $tp$; a control $c$ of a $c$-specific
             TPF collection $F$
       Output: The next mapping $\mu'$ such that $\mu' \in [\![\{tp\}]\!]_F$, or nil when no
               such mappings are left
2      self.$\phi$ ← an empty page without triples or controls if self.$\phi$ is
         unassigned;
3      while self.$\phi$ is empty or only contains read triples do
4          if self.$\phi$ has a control to a next page with URL $u_{\phi'}$ then
5              self.$\phi$ ← GET $u_{\phi'}$;
6          else
7              self.$\mu_s$ ← $I_s$.GetNext();
8              return nil if self.$\mu_s$ = nil;
9              self.$\phi$ ← GET $c(\{$self.$\mu_s[tp]\})$, resulting in page 1 of the TPF
                 for self.$\mu_s[tp]$;
10         end
11     end
12     $t$ ← an unread triple from self.$\phi$;
13     $\mu$ ← a solution mapping $\mu'$ such that $\mathrm{dom}(\mu') = \mathrm{vars}(tp)$ and $\mu'[tp] = t$;
14     return $\mu \cup$ self.$\mu_s$;
15 end
```

Listing 4.3: A `TriplePatternIterator` incrementally evaluates a triple pattern *tp* over a *c*-specific TPF collection *F*.

Finally, a `BasicGraphPatternIterator` combines the above two iterators to evaluate BGPs. For an empty BGP, the `BasicGraphPatternIterator` constructor creates a `RootIterator` instead (as the corresponding query result contains only the empty mapping $\mu_\emptyset$); for a singleton BGP, a `TriplePatternIterator` is constructed (as no further decomposition is needed). In all other cases, Listing 4.4 is executed.

```
1  Function BasicGraphPatternIterator.GetNext()
       Data: A source iterator $I_s$; a BGP $B$ with $|B| \geq 2$; a control $c$ of a $c$-specific
             TPF collection $F$
       Output: The next mapping $\mu'$ such that $\mu' \in [\![B]\!]_F$, or nil when no such
               mappings are left
2      $\mu \leftarrow$ nil;
3      self.$I_p \leftarrow$ nil if self.$I_p$ is unassigned;
4      while $\mu =$ nil do
5          while self.$I_p =$ nil do
6              self.$\mu_s \leftarrow I_s$.GetNext();
7              return nil if self.$\mu_s =$ nil;
8              foreach triple pattern $tp_j \in B$ do
9                  $\phi_1^j = \langle u_1^j, u_j, s, \Gamma_1^j, M_1^j, C_1^j \rangle \leftarrow$ GET $c(\{$self.$\mu_s[tp_j]\})$, resulting in
                     page 1 of that TPF;
10                 $cnt_j \leftarrow cnt$ where $\langle u_j, \texttt{void:triples}, cnt \rangle \in M_1^j$;
11             end
12             if $\forall j : cnt_j > 0$ then
13                 $\epsilon \leftarrow j$ such that $cnt_j \leq cnt_k \; \forall tp_k \in B$;
14                 $I_\epsilon \leftarrow$ TriplePatternIterator(RootIterator(), self.$\mu_s[tp_\epsilon]$, $c$);
15                 self.$I_p \leftarrow$ BasicGraphPatternIterator($I_\epsilon$, self.$\mu_s[B \setminus \{tp_\epsilon\}]$, $c$);
16             end
17         end
18         $\mu \leftarrow$ self.$I_p$.GetNext();
19         self.$I_p \leftarrow$ nil if $\mu =$ nil;
20     end
21     return $\mu \cup$ self.$\mu_s$;
22 end
```

Listing 4.4: A BasicGraphPatternIterator incrementally evaluates a BGP B over a c-specific TPF collection F.

The main principle of a BasicGraphPatternIterator is that it creates a *separate* iterator pipeline for each incoming solution mapping. The iterator has two member variables: self.I_p to hold the current iterator pipeline, and self.μ_s to hold the most recently read mapping from its source iterator I_s. Upon reading a mapping self.μ_s, a BasicGraphPatternIterator identifies which of the triple patterns in $\{$self.$\mu_s[tp_j] \mid tp_j \in B\}$ has the lowest estimated total matches by fetching the first pages of the corresponding TPFs (lines 8 to 13). Then, a new iterator pipeline self.I_p is created, consisting of a TriplePatternIterator for the identified triple pattern and a BasicGraphPatternIterator for the remainder of B (lines 14 to 15).

This algorithm is *greedy* causing some join cases to lead to a Cartesian product. Early improvements by Van Herwegen et al. [23] and Acosta and Vidal [24] propose global decisions in favor of local ones at higher client cost.

The mappings μ returned by this pipeline are then combined with the input mapping `self`.μ_s and returned (lines 18 to 21). In other words, the BGP is evaluated by splitting off the “simplest” triple pattern at each stage. For Listing 4.2, the `BasicGraphPatternIterator` would thus split off tp_3, and create a pipeline for $\{tp_1, tp'_2\}$. This process is *dynamic*: instead of constructing a static pipeline for a BGP upfront, a local pipeline is decided at each step.

Note in particular that only `TriplePatternIterators` read more than one page of a TPF. `BasicGraphPatternIterators` only fetch first pages, and never read actual data, only metadata. In an efficient implementation, pages are cached locally, so that the `TriplePatternIterator` need not fetch the first page again—and in general, no page should be fetched more than once.

4.3 Evaluation of the TPF interface

This client is written in JavaScript, so it can be used either as a standalone application, or as a library for browser and server applications. We provide all source code of the implementations, as well as the full benchmark configuration at https://github.com/LinkedDataFragments.

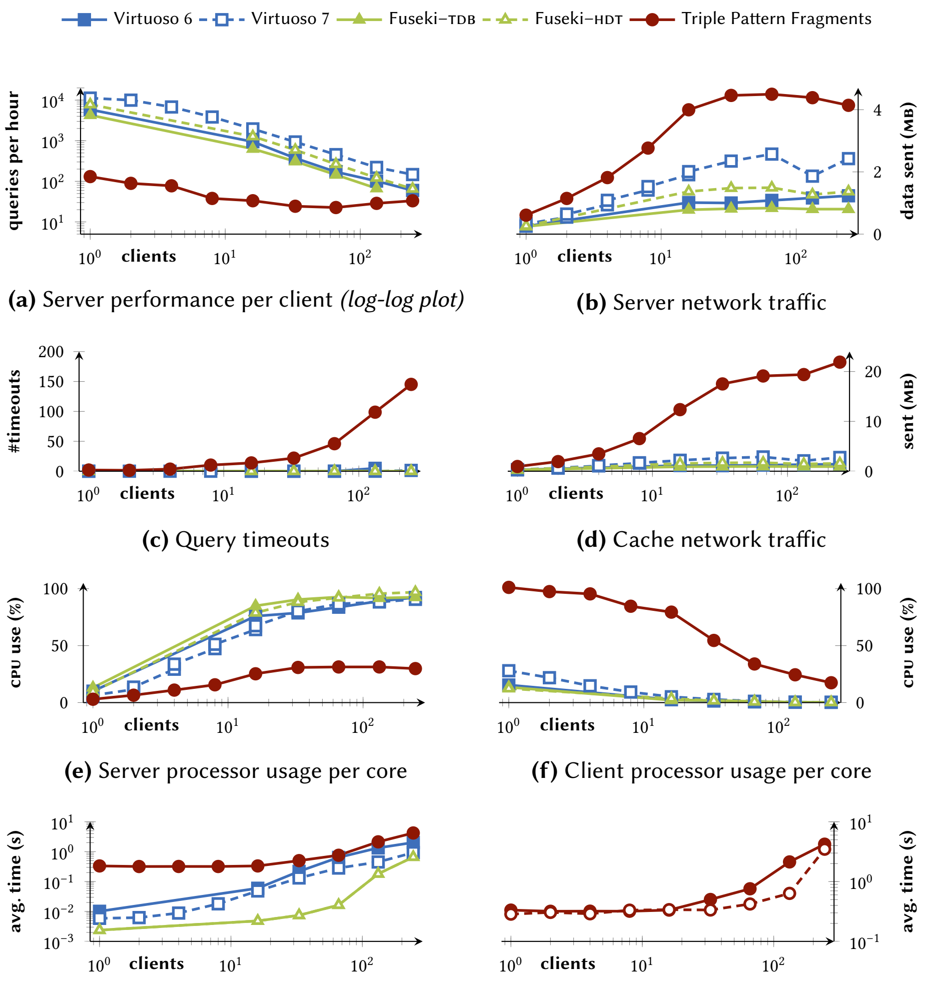
This evaluation aims to compare the relationship between server cost and performance of triple-pattern-based query execution to query execution over other LDFs, SPARQL endpoints in particular. As a basis for the experiments, we implemented the query execution approach from Section 4.2 as an open-source LDF client for SPARQL queries.

4.3.1 Influence of client numbers

A first experiment should indicate whether TPF reduces server-side costs compared to SPARQL endpoints, hence being a more scalable alternative to publish Linked Data in queryable form. We measure cost as CPU usage, the cache hit ratio, and the average response time. The usage of RAM and I/O highly depend on the implementation of the data source and are therefore not included directly. With scalability, we mean its sustainability under an increasing number of clients that concurrently access the interface. A maximum of 240 simultaneous clients was set, as any higher number would measure an unrealistic scenario.

Experimental setup

In this experiment, we compare a TPF client/server setup to four SPARQL endpoint based setups. For the latter we use Virtuoso (6.1.8 and 7.1.1) [25] and Jena Fuseki [26] (TDB 1.0.1 and HDT 1.1.1), respectively; and for the TPF server we use an HDT [9] backend. In all cases, the HDT file was memory-mapped and thus not entirely available in the memory. Virtuoso was configured with the recommended settings `NumberOfBuffers` = 595000, `MaxDirtyBuffers` = 455000, and `MaxCheckintRemap` as 1/4 of `NumberOfBuffers`.



(a) Server performance per client *(log-log plot)*

(b) Server network traffic

(c) Query timeouts

(d) Cache network traffic

(e) Server processor usage per core

(f) Client processor usage per core

(g) Query 3 execution time *(log-log plot)*

(h) Query 3 execution and 1st solution time (dotted) *(log-log plot)*

Figure 4.2: Main measurements on the influence of client numbers

To measure the cost and performance of the TPF server and the SPARQL endpoints under varying loads, we set up an environment with one server and a variable number of clients on the Amazon AWS platform. The complete setup consists of 1 server (4 virtual CPUs, 7.5 GB RAM), 1 HTTP cache (8 virtual CPUs, 15 GB RAM) and 60 client machines (4 virtual CPUs, 7.5 GB RAM), capable of running 4 single-threaded clients each. All machines have Intel Xeon E5-2666 processors running at 2.60 GHz. The HTTP cache acts as a proxy server between servers and clients and was chosen for its bandwidth capabilities (identified by Amazon with specific CPU/RAM combinations). The maximum lifetime of cached resources is set to 5 mins, with a maximum of 1000 simultaneously cached documents that are replaced according to the Last-Recently-Used strategy.

As a benchmark for the experiment, we chose the Berlin SPARQL Benchmark (BSBM) [27], for the following reasons:

- the BSBM was designed to compare SPARQL endpoints across architectures [27], and we aim to compare our client–server architecture to such traditional server architectures;
- the BSBM aims to simulate realistic workloads with large amounts of RDF data [27];
- the BSBM contains parameterized queries to create different workloads for large numbers of clients.

In particular, we used a BSBM instance with a dataset size of 100M triples. To mimic the variability of real-world scenarios, each client executes different BSBM query workloads based on its own random seed. Some of the BSBM queries use the `ORDER BY` operator, which our TPF client necessarily implements with a blocking iterator, so the first solution can only be output after all solutions have been computed. Therefore, (only) for measurements of the first solution time we use variants of these queries without `ORDER BY`, assuming the user application prefers streaming results and sorts locally. After every 1-second interval during the evaluation, we measure on the server, cache, and clients the current value of several properties, including CPU usage of each core and network I/O. For every tested number of clients, the HTTP cache starts cold and is not reset during query workload execution.

Results

Figures 4.2a to 4.2h summarize the main measurements of the evaluation. All x-axes use a logarithmic scale, because we varied the number of clients exponentially. We measured all data points for Virtuoso 7 (as latest and best performing version) and our proposed solution. For the other alternatives, we measured the points most relevant for the analyses. Figure 4.2a shows that the per-client performance of SPARQL endpoints decreases significantly with the number of clients. Even though a TPF setup executes SPARQL queries with lower performance, the performance decrease with a higher number of clients is significantly lower. Because of caching effects, TPF querying starts performing slightly better with a high number of clients ($n > 100$). The per-core processor usage of the SPARQL endpoints grows rapidly (Figure 4.2e) and quickly reaches the maximum; in practice, this means the endpoint spends all CPU time processing queries while newly incoming requests are queued. The TPF server consumes only limited CPU, because each individual request is simple to answer, and due to the coarser granularity of the selector function, the cache can answer several requests (Figure 4.2d).

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>

SELECT ?product ?label
WHERE {
 ?product rdfs:label ?label .
 ?product a \%ProductType\% .
        ?product bsbm:productFeature \%ProductFeature1\% .
        ?product bsbm:productPropertyNumeric1 ?p1 .
        FILTER ( ?p1 > \%x\% )
        ?product bsbm:productPropertyNumeric3 ?p3 .
        FILTER (?p3 < \%y\% )
 OPTIONAL {
 ?product bsbm:productFeature \%ProductFeature2\% .
 ?product rdfs:label ?testVar }
 FILTER (!bound(?testVar))
}
ORDER BY ?label
LIMIT 10
```

Listing 4.5: BSBM query template 3: finding products that satisfy 2 numerical inequalities and an OPTIONAL clause (%...% indicates a parameter that is replaced at runtime according to the current query workload).

At the client side, we observe the opposite (Figure 4.2f): clients of SPARQL endpoints hardly use CPU, whereas TPF clients use between 20% to 100% CPU. With an increasing number of concurrent TPF clients, the networking time dominates and, thus, the per-client CPU usage decreases, whereas the memory consumption does not vary significantly (ca. 0.5 GB per client, and 8 GB on the server).

Figure 4.2b shows the outbound network traffic on the server with an increasing number of clients. This traffic is substantially higher for the TPF server, because TPF clients need to perform several requests to evaluate a single query. The cache ensures that responses to identical requests are reused; Figure 4.2d shows that caching is far more effective with TPFs.

Some of the BSBM queries perform comparably slow on TPF clients, especially those queries that depend on operators such as FILTER, which in a triple-pattern-based interface can only be evaluated on the client. The execution times of some of these queries exceed the timeout limit of 60s (Figure 4.2c). A "timeout" means that query execution was stopped before *all* results arrived; at least a partial result set was already computed. An example of such queries are instances of BSBM query template 3 given in Listing 4.5. Focusing only on the queries of this template, we observe that the average execution time of these queries is comparably higher for TPFs (Figure 4.2g). However, with an increasing number of clients, these times increase only very gradually in the TPF setup, whereas they rise very rapidly for the SPARQL endpoints (which have to compute the queries of all clients concurrently). Furthermore, the time to the first solution increases more slowly with increased load (Figure 4.2h). Only on

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT * WHERE {
 { ?v2 a dbo:Settlement; rdfs:label \%\%v\%\%.
   ?v6 a dbo:Airport. }
 { ?v6 dbo:city ?v2. }
 UNION { ?v6 dbo:location ?v2. }
 { ?v6 dbp:iata ?v5. }
 UNION { ?v6 dbo:iataLocationIdentifier ?v5. }
 OPTIONAL { ?v6 foaf:homepage ?v7. }
 OPTIONAL {?v6 dbp:nativename ?v8. }
}
```

Listing 4.6: Example query from DBpedia benchmark: finding airport information of a certain city with two UNION clauses and an OPTIONAL clause (%%...%% indicates a parameter that is replaced at runtime according to the current query workload).

the TPF server, CPU usage remains low for this query at all times.

4.3.2 Performance of queries on a real-world dataset

Our second experiment aims to extend the results of the previous experiment toward real-world datasets and different dataset sizes. It is essential to ascertain whether the majority of typical real-world queries can be executed within the acceptable limit of four seconds [28].

Experimental setup

The experimental setup is the same as before. However, this time, we want to validate the behavior of the TPF client for increasing real-world dataset sizes. To this end, we execute the DBpedia SPARQL benchmark [29], which uses a real-world dataset. The benchmark incorporates queries from the public DBpedia SPARQL endpoint log, filtering non-relevant variations and queries with a low frequency [29]. We use three dataset sizes as made available online: 14,274,065 triples and 52,323,498 triples from the DBpedia benchmark website[1], and 377,367,913 triples from the 2014 version of DBpedia (without long abstracts). In contrast to the BSBM we used in the last experiment, the DBpedia query logs do not contain parameters. Therefore, clients execute a mix of 15 queries in sequence with different template values for each client. Listing 4.6 shows an example such query template. Query timeout was set to 60s.

1. http://benchmark.dbpedia.org/

	14M	52M	377M
Q1	0.44	0.42	0.37
Q2	0.50	0.50	0.46
Q3	0.50	0.51	0.91
Q4	0.53	0.60	1.49
Q5	0.47	0.48	0.86
Q6	1.80	10.06	35.16
Q7	0.47	0.46	0.40
Q8	0.50	0.49	0.46
Q9	0.47	0.45	0.39
Q10	0.47	0.45	0.46
Q11	0.52	0.92	0.43
Q12	60.03	60.03	59.95
Q13	2.29	3.16	21.80
Q14	0.48	0.47	0.43
Q15	0.73	1.08	7.69
Total	70.19	80.09	131.27

Table 4.1: Average individual query execution time (in seconds) for the three DBpedia dataset sizes with 48 concurrent TPF clients

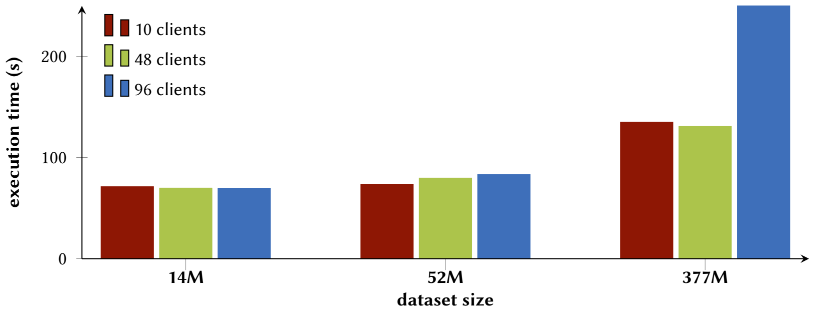


Figure 4.3: Average execution times per client of a DBpedia query mix with variable dataset sizes

Results

Figure 4.3 shows the results for 10, 48, and 96 TPF clients. We observe an increase in query execution time, which is minor when going from the 14M graph to the 52M graph, but more visible for the entire version of DBpedia 2014. The main cause for this increase is the increased number of elements in the result set, as the total number of triples influences the number of matching triples per query. To a lesser degree, the execution times are also influenced by an increased number of triples that match one pattern but cannot be used in joins.

The most important observation in this experiment is, however, the high variance in execution time across queries. Table 4.1 shows the average query execution times for an individual client per query, for the case of 48 concurrent clients. Note how Q12 reaches the 60s timeout even for the small dataset. This is due to the presence of various `UNION`, `OPTIONAL`, and `FILTER` constructs, for which our client does not generate efficient query plans. Most of the query execution times remain at the same magnitude for the different dataset sizes, with small differences accounting for factors such as caching and increasing bandwidth consumption. Queries Q6, Q13 and Q15 are affected more clearly by increasing dataset sizes; at the same time, however, they also yield more results. This indicates that, more than dataset size, the type of query seems to be a crucial factor for queries against real-world datasets such as DBpedia.

4.3.3 Influence of serialization formats

The delay to transfer triples from the server into a memory-based representation of the client is much more crucial for TPF clients than for SPARQL endpoints, because the number of required HTTP requests per query can be high. Therefore, we study the impact of the RDF serialization format of TPF responses on SPARQL query execution performance. The serialization format contributes to the delay in two ways: *(a) response processing delay*, i.e., coding and decoding triples, and *(b) response download delay*, i.e., transporting the response over a network. A serialization format offers a specific mix of both, which can be more or less suited for a certain Web application. We evaluated different existing formats to discover a balanced mix between processing and download cost in the context of our client-side querying algorithm. Subsequently, we can obtain recommendations on *(a)* characteristics for new serialization formats, and *(b)* server page size configurations.

Serialization formats

To test with a diverse mix of formats we identified three categories—text-based, processing-oriented binary, and download-oriented binary—from which we selected an overall number of twelve different formats, and used each of them with and without additional GZIP compression, which is common with HTTP.

Text-based formats are string-encoded notations aimed at both human and machine consumption. We selected N-Triples [30] and its superset Turtle [31] mentioned in Section 2.3. The Apache Jena RIOT library[2] was used for both formats, using stream processing when possible. For Turtle, we tested the three configurations *pretty* (pre-sorted by subject with maximal grouping; non streaming), *flat* (not pre-sorted and not grouped; triple-based streaming), and *block* (pre-sorted by subject and grouped in fixed windows; window-based streaming). In addition, we tested with the Sesame RIO[3] library to include a possible implementation difference.

2. https://jena.apache.org/documentation/io/
3. http://rdf4j.org/sesame/2.8/docs/programming.docbook?view#Parsing_and_Writing_RDF_with_Rio

Processing-oriented binary formats are binary notations optimized to reduce processing delay, which often comes with a response size penalty. Fortunately, such formats tend to work well with common compression, e.g., GZIP. This compensates response size increase at the cost of processing time. In this category, we selected RDF Thrift [32], implemented by Jena RIOT, and Sesame Binary RDF [33], implemented by Sesame RIO. For RDF Thrift we tested both the default configuration and the *Values* configuration, where literals are encoded more efficiently. For Sesame Binary we used both a buffer size of 800 and of 1,600 triples.

Download-oriented binary formats are optimized to reduce download delay. They offer compression techniques that greatly reduce response size, often sacrificing processing time. We added the recent ERI format [34] using a custom implementation[4] by the authors. We tested with the recommended block sizes of 1,024, 2,048, and 4,096 triples.

Experimental setup

To obtain collections of TPF responses for which we could measure the impact of each of the selected serialization formats, we instructed our TPF client implementation to store each TPF page retrieved during a query execution as a local RDF file. Then, to take the measurements for this experiment we implemented a single-threaded Java application that loads such a local RDF file into main memory and serializes and deserializes the loaded data using the different serialization formats mentioned before. This application was deployed on an Intel Xeon CPU (E5-2640 2.50 GHz) with 1 TB HDD RAID storage and 64 GB DDR3 1333 MHz RAM.

To ensure diversity of the collections of TPF responses as considered for this experiment, we used queries and a dataset of the Waterloo SPARQL Diversity Test Suite (WatDiv) [35]. In contrast to the previous two experiments, which focused on realistic loads, we here specifically want to assess the impact of different query patterns, which is where WatDiv was designed for. We set up our TPF server with the WatDiv 10 million triple dataset[5] and queried it using 20 BGP queries that we generated from the 20 query templates in the basic testing use case[6] of WatDiv. These query templates span a wide spectrum of various graph pattern structures (cf. Table 4.2). For each of the 20 queries, Table 4.2 shows the number of TPF pages requested by our client-side query execution algorithm.

The 20 WatDiv templates are graphically displayed at http://db.uwaterloo.ca/watdiv/basic-testing.shtml. Note that the number of templates per category does not necessarily reflects actual query distributions for specific datasets.

4. https://github.com/webdata/ERI
5. http://dsg.uwaterloo.ca/watdiv/watdiv.10M.tar.bz2
6. http://dsg.uwaterloo.ca/watdiv/basic-testing.shtml

template name	**triple patterns**	**variables**	**join vertices**	***TPF pages***
L1	3	3	2	*9,159*
L2	3	2	2	*14,815*
L3	2	2	1	*59,062*
L4	2	2	1	*357*
L5	3	3	2	*315*
F1	6	5	2	*900*
F2	8	8	2	*184*
F3	6	6	2	*1,793*
F4	9	8	2	*869*
F5	6	6	2	*216*
S1	9	9	1	*251*
S2	4	3	1	*1,415*
S3	4	4	1	*455*
S4	4	3	1	*67*
S5	4	3	1	*304*
S6	3	3	1	*33*
S7	3	3	1	*594*
C1	8	9	4	*869*
C2	10	11	6	*431*
C3	6	7	1	*4*

Table 4.2: Properties of our WatDiv queries (template from which the query is generated, number of triple patterns, number of variables, join vertex count [35]), and the number of TPF pages requested during their execution. WatDiv distinguishes linear queries (**L**), snowflake-shaped queries (**F**), star queries (**S**), and complex queries (**C**).

Results

First, we analyze how much data the collected TPF pages contain (which is independent of serialization formats). Figure 4.4 shows a distribution of the number of triples per page. The histogram starts with 25 triples, because a response always includes at least 26 triples for metadata and controls, and ends with 130 triples because a page size of 100 triples was used. All queries show a similar bimodal distribution with a peak at a triple count of 25 to 35 triples and a small peak at 125 to 130 triples. The former indicates the presence of many requests that are used to verify the presence of a single triple, which results in 0 or 1 data triples. The latter denotes the initial phase of query execution where many non-selective triple patterns are requested. Most frequent, these are the triple patterns as they occur in the query's BGP and some of their early bound derivatives. These patterns contain the highest number of variables, more likely to exceed the given page size. Examples are the triple patterns tp_1 and tp_2 from the query in Listing 4.2, that have a cardinality of 2,300 and 572,300, respectively. On average, only 2% of all responses contain between 35 and 125 triples. This means HTTP responses are in most cases very small, in a few cases very big (depending on the page size), and almost never in-between. Two queries (L2 and L3) have more fragment pages in the in-between groups, which is directly related to the fact they have more pages overall.

Next, we executed the response serializer for each combination of query and format, with and without applied GZIP compression. For each response, we measured its serialization time, deserialization time, and transfer time. We estimate the transfer time by dividing the response size when serialized with an average network speed of 1MB/s (as in other evaluations [34]). The sum

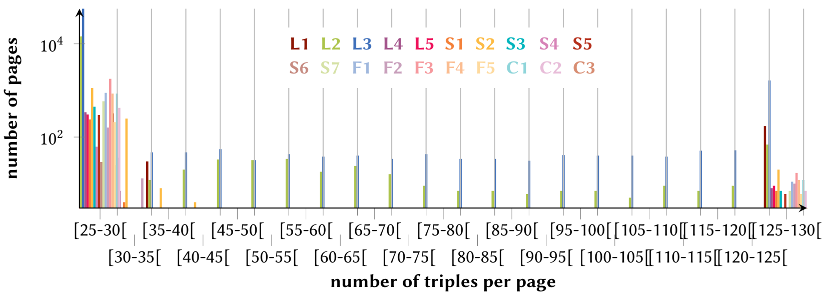


Figure 4.4: The distribution of average number of triples per used TPF page for different queries shows that most fragments either contain few data triples (20–30 triples are metadata), or the maximum number (page size).

of all three measurements over all HTTP responses give the total overhead per query.

Figure 4.5 shows the average overhead over all queries. A first observation is that transfer time clearly dominates over serialization and deserialization time. Therefore, the small-size ERI format results in roughly 3 times less overhead compared to Turtle and even 5 times less compared to N-Triples, RDF Thrift, and Sesame Binary. However, since text-based formats and processing-oriented formats can be compressed effectively, the response size difference can be reduced significantly by applying GZIP compression. On average, text-based formats can be reduced to 110% of the ERI response size. Nonetheless, the processing penalty by applying compression does not compensate the dominance of transfer delay for most formats, as ERI keeps an average gain of 1s per query. For N-Triples though, the combination of its high compression rate and lower processing cost match the low overhead of ERI.

In general, the impact of binary formats is rather limited. This is likely because TPF pages are small, while such formats are designed to be effective on large amounts of triples. All responses are serialized as a single block or buffer, which explains the insignificant difference between block sizes for ERI and Sesame Binary. A larger page size could result in a potential overhead decrease. However, for all tested queries, only 0.24% of all fragments requested more than one page and only 0.001% more than two. As a result, the benefit of a larger page size would be negligible in the current algorithm.

Future algorithms could justify larger page sizes by changing the distribution of page requests. For instance, Van Herwegen et al. [23] reduce the number of requests by moving the join order optimization from local to global. Based on multiple heuristics, the client estimates from the intermediate results whether the approach from Section 4.2.3 is suboptimal, and if so, downloads entire triple patterns separately instead, i.e., downloading all pages of a single TPF at once and performing a symmetric hash join [36]. Also, the planned migration of the Web to HTTP/2 [37], with features such as *multiplexing* (i.e., using a single con-

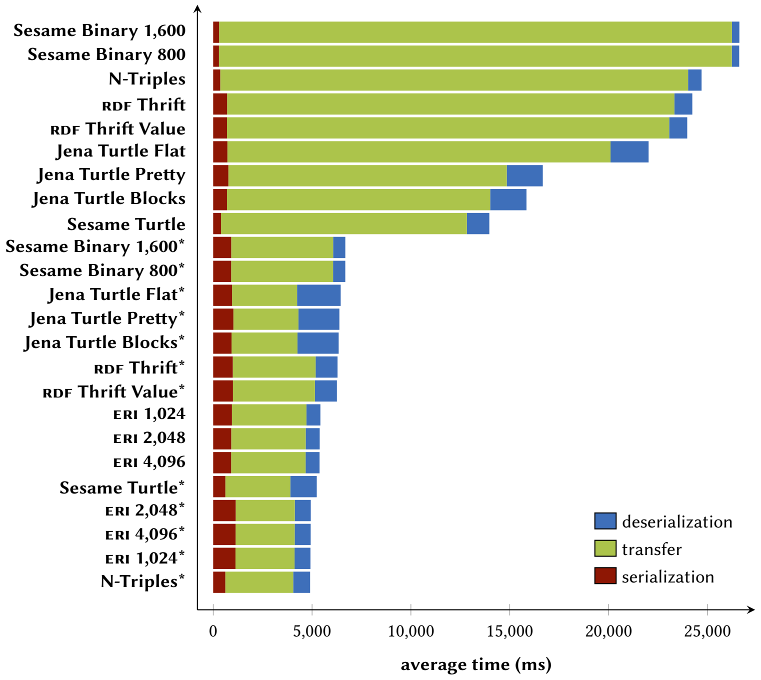


Figure 4.5: The chosen RDF format impacts the time to serialize, transfer, and deserialize TPF pages. Additionally applying GZIP compression and decompression (indicated by *) has an important effect, except on ERI, which is already small.

nection to sent multiple HTTP requests and receive responses asynchronously) and *push* (i.e., a server proactively sends multiple responses for a single request), could change the circumstances in which the size of a page is considered optimal.

4.4 Adding Approximate Membership metadata

As the evaluation in Section 4.3 suggests, the required time for a client to evaluate certain SPARQL queries against TPF interfaces can still be unacceptable for responsive applications. A dominant factor this time is the high number of HTTP requests. Therefore, by analyzing the nature of these requests, we can locate possible areas for changing the client/server trade-offs in the interface. To this end, we analyze the page requests from Table 4.2, which result from executing the WatDiv benchmark [35] against a TPF interface using the algorithm from Section 4.4.1.

The execution logs revealed a high number of requests for triple patterns without variables, i.e., testing the presence of a specific triple in the dataset. Such a request is called a *membership subquery*.

Definition 12 (Membership subquery). A membership subquery is a triple pattern query without variables that effectively checks the membership of a triple in the dataset.

The templates L2, L4, and F3 respectively produced 50%, 51%, and 74% membership subqueries. For S5, F5, C1, and C2, this proportion even reached 95% to 98%. Furthermore, the absolute number of requests of some of these templates is high (e.g., F3 needed 1,335 membership subqueries). A third of query templates is thus affected; the remaining 13 templates produced no membership subqueries at all. While these numbers do not allow generalized conclusions, they are certainly an important indication that a reduction of membership subqueries can have a considerable influence on the number of HTTP requests—and thus the overall query execution time.

Therefore, we augmented the metadata of the TPF interface with *approximate membership functions* (AMF). An AMF is a space-efficient data structure that is able to indicate whether a set contains an item [38]. False positives can occur with a fixed probability, while false negatives cannot occur. We study their applicability as a server-side feature in addition to TPF—can they reduce the number of HTTP requests during client-side SPARQL query execution, while maintaining a low per-request cost for the server?

4.4.1 Approximate membership functions

In the following, we briefly discuss two prominent AMF families—Bloom filters [39] and Golomb-Coded Sets (GCS) [40]—and some existing work from literature concerning RDF querying using AMF.

Bloom filters and GCS both offer approximate membership assessment with a predefined false positive probability, but with different size and speed. Recall and precision are important parameters of an AMF f. Given the set of actual members M and a set of elements T for which we want to test membership, the set of positively tested elements $P_T = \{t \in T : f(t) = \texttt{true}\}$. We define $recall_f(T) = |M \cap P_T|/|M|$ and $precision_f(T) = |M \cap P_T|/|P_T|$. Both Bloom filters and Golomb-Coded Sets have 100% recall, i.e., all valid members of M will always be identified, but less than 100% precision.

Bloom filters

A Bloom filter [39] is a bitmap of m bits populated using k different hash functions, initialized with all bits set to 0. An item is added by calculating k locations in the bitmap, which are set to 1. Each one is calculated by using a different hash function to ensure randomness. An item can be tested by calculating k locations using the same hash functions. Hence, both insertion and testing are $O(k)$. The result of a bit-wise `AND` of those locations in the filter determines if the item is a member. If `false`, the item is *definitely* not in the set. If `true`, the item *might* be in the set, because of false positives.

For a desired false positive probability rate p, the bit-size of a Bloom filter is proportional to its number of members n. The required size is $m = -n \cdot \log_2 e \cdot \log_2 p$. For a given m, the optimal number of hashes k that minimizes false positive probability can be calculated with $k = m/n \cdot \ln 2$. Despite their compact representation, their size can be too large for network transfer. A solution is using compressed Bloom filters [41], at the cost of compression and decompression delays.

Golomb-coded sets

Golomb-coded sets (GCS) [40] provide a cleaner variation of compressed Bloom filters. First, only a single hash function is used to populate a bitmap with. For n members, this results in a bitmap of n bits. Then, the locations of the bitmap that are set to 1, compose a list of indexes l, which is considered to be uniformly distributed. Next, the differences between all subsequent values of l form a new list l', which has a geometrical distribution with a parameter p. Finally, a GCS is created by compressing l' with golomb-coding, which is an optimal encoding for discrete geometric distributions [42].

In terms of size, GCS approaches the theoretical minimum of $m = -n \cdot \log_2 p$ more closely than the equivalent Bloom filter. Compared to compressed Bloom filters, GCS have a minimal size overhead for the same p, but they are more easily chunked and indexed to deal with uncompressed size issues. Compared to plain Bloom filters, the query time is magnitudes slower due to decompression. However, this drawback can be minimized by including an index to quickly find areas of interest in the filter.

Query evaluation with approximate membership

In the context of RDF querying, approximate membership functions are included in several related works, covering *(a)* query routing in networks, *(b)* selectivity estimation for optimizing joins, *(c)* evolutionary querying, and *(d)* local database indexes.

Query routing applies Bloom filters in caches and indexes for peer-to-peer, MapReduce or cloud clusters, and Linked Data networks. Most systems [43, 44, 45] construct a *data summary* of neighboring nodes or clusters to make query forwarding decisions. Some algorithms exchange these filters between nodes to maintain their network [46]. This is common in combination with Distributed Hash Tables (DHT) [46, 47], where a DHT is used for data routing and Bloom filters for efficient communication between nodes.

More directly applicable is *selectivity estimation* of query patterns, e.g., graph patterns, to improve join performance. One approach is to group different chain-patterns, i.e., two distinct triple patterns connected by a single variable, according to their frequency [48]. A Bloom filter tests in what frequency group a chain pattern resides, which optimizes the pattern execution order. Other applications include representing equivalent classes to optimize hash joins, ranges of values for merge joins [49], and distributed n-way joins [50]. Although these works inspire future directions, many require more than a single triple pattern and have high demands for the server. Highly relevant is the proposal to extend the ASK query response [51] with combinations of bindings, i.e., two variables in a triple pattern, to improve source selection in SPARQL query federation frameworks. Bloom filters from different sources indicate overlap and save redundant requests. However, the benefit in a single-server setup is unclear.

Evolutionary querying is an alternative approach to SPARQL query processing. Possible solutions are first gathered as an initial population, e.g., the entire dataset, which then mutate and evolve to be refined incrementally. Oren, Guéret, and Schlobach [52] use a combination of fingerprinting and Bloom filters to rapidly evaluate approximate answers against large RDF datasets. Although this is a centralized solution, it advocates *anytime* answers, which is in line with the opportunistic querying presented in this paper. The algorithm is initiated with random values, which returns initial results fast, but with low accuracy.

Finally, in the area of databases, Bloom filters are an efficient technique to prevent unnecessary disk access [53]. In such cases, the size of the filter and its impact on transfer delays are not applicable.

4.4.2 Definition and Response format

The self-descriptive nature of the TPF interface allows us to add AMFs as extra server feature without any interference, as long as we can describe them. Therefore, we created the membership modeling ontology shown in Figure 4.6 (henceforth denoted with the prefix `ms`). To differentiate the impact on response size, we consider two different AMF techniques discussed earlier: the uncompressed Bloom filters [39] and the compressed Golomb-Coded Sets (GCS) [40].

The membership ontology defines `ms:Function` for generic functions and its subclasses `ms:ApproximateMembershipFunction` and `ms:HashFunction`. To allow for Bloom filters and GCS, the former has `ms:BloomFilter` and `ms:GolombCodedSet` as subclasses. Finally, `ms:hashFunction` associates instances of these classes with hash functions that can be instances of algorithms such as `ms:MD5` or `ms:MurmurHash3`. Using this ontology, we define an interface feature that provides AMF metadata in the metadata graph of responses. The result is the subject of a specification in the Hydra W3C Community Group [54].

As mentioned in Section 4.1, each regular TPF contains a `void:triples` statement expressing the approximate total number of triples in the dataset that match the TPF's triple pattern [11]. For instance, each page of the TPF for the

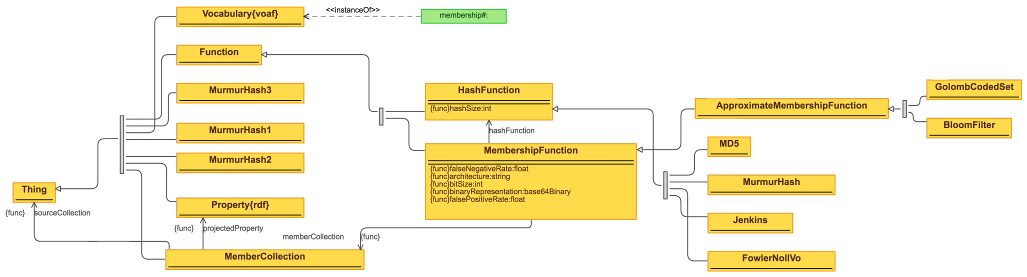


Figure 4.6: The membership modeling ontology published and maintained at http://semweb.mmlab.be/ns/membership.

pattern "`?x rdf:type foaf:Person`" contains a metadata triple stating there are 96,300 matching triples in the dataset. Given a page size of 100 data triples, these data triples would be spread across 963 pages. Suppose that during the execution of a certain SPARQL query, the client arrives at a list of 215 potential mappings for "`?x rdf:type foaf:Person`". In order to verify with a minimum number of HTTP requests whether these mappings are valid, the 215 TPFs for the corresponding triples need to be downloaded, checking which mappings result in a triple that exists within the dataset.

By defining an interface feature that allows this fragment to contain an AMF, the clients can determine approximately whether a certain `?x` results in a triple of the dataset. Thus, the properties of the LDFs, to which this enhanced TPF interface gives access to, become:

data & controls: equivalent to TPF;
metadata: equivalent to TPF plus an AMF per variable in the triple pattern representing the variable's set of matching URIs or literals.

The hash functions themselves are not detailed in the listing, but their parameters need to be explicitly available (either in the response or by dereferencing their URL).

Listing 4.7 shows an example AMF for the triple pattern "`?x rdf:type foaf:Person`". In this case, it is a Bloom filter (line 4) with two specific Murmur functions as hash functions (line 6). Lines 7 to 10 explicitly specifies that the members of the collection are the triples of the fragment, and that the AMF has been built by using the *subject* of these triples. This allows the client to interpret how exactly this AMF can be used. For instance, if the triple `dbp:Elvis_Presley rdf:type dbo:Artist` is part of the dataset, then the full URI of `dbp:Elvis_Presley` must yield a positive value in the membership function. Note that the false positive rate and the possible false negative rate are also specified in lines 11 and 12, allowing a client to estimate the certainty of each result. Finally, the AMF data itself has been made available in base64-encoded form (line 13).

This metadata allows a client to unambiguously recreate the AMF and verify

```
<#metadata> foaf:primaryTopic <#fragment>.
<#metadata> {
   <#fragment> void:triples 96300.         # existing count metadata
   _:membershipFunction a ms:BloomFilter;  # AMF metadata
      ms:hashSize 524288;
      ms:hashFunction <MyMurmur1>, <MyMurmur2>;
      ms:memberCollection [
         ms:sourceCollection <#fragment>;
         ms:projectedProperty rdf:subject
      ];
      ms:falsePositiveRate 0.05;
      ms:falseNegativeRate 0.0;
      ms:binaryRepresentation "QmF...ZTY"^^xsd:base64Binary.
}
```

Listing 4.7: The self-descriptive AMF metadata in the TPF fragment for `?x rdf:type foaf:Person` allows the client to interpret and evaluate approximate membership.

the approximate membership of elements. Note that this self-descriptive approach does not require a contract between the client and the server, e.g., no hash function has to be agreed upon silently. Furthermore, clients that do not use this metadata feature, such as the TPF client from Section 4.2, will not be affected by it and can thus continue to use the interface. It is up to the server's discretion whether or not to provide an AMF on a page. If it is present, an AMF-aware client can use it; if not, the original algorithm without AMFs can be followed. This lets the server choose freely what metadata to include—based on, for instance, the computational effort to create the AMF.

4.4.3 Querying an AMF-enabled TPF interface with SPARQL

To query TPFs with AMF metadata, we will add a small extension to the algorithm presented in Section 4.2. Consider the following example query for DBpedia:

```
SELECT ?p ?c WHERE {
 ?p a <http://dbpedia.org/ontology/Artist>.                  # tp1
 ?p <http://dbpedia.org/ontology/birthPlace> ?c.             # tp2
 ?c <http://www.w3.org/2000/01/rdf-schema#label> "York"@en.  # tp3
}
```

Listing 4.8: This SPARQL query finds artists born in cities named "York".

Given a regular TPF interface, the current algorithm will compute results for each BGP B by recursively evaluating and binding each triple pattern $tp_i \in B$ in an order determined by the count metadata in their respective fragments. For example, by fetching the first page of the TPFs for the query in Listing 4.8 where $B = \{tp_1, tp_2, tp_3\}$, we obtain the count metadata $\{(tp_1, 96\,300), (tp_2, 625\,811), (tp_3, 2)\}$. Therefore, we start iterating over tp_3, which will supply values for ?c. This leads to 2 subqueries $B' = \{tp_1, tp_2'\}$ where the remaining triple patterns are bound to concrete values of ?c (note that tp_1 is unaffected because it does not contain `?c`). For instance, for `?c = dbp:York`, we obtain count metadata $\{(tp_1, 96\,300), (tp'_2, 207)\}$. Query execution thus continues with the smallest fragment tp'_2, which results in 207 subqueries $B'' = \{tp'_1\}$ in which tp_1 is bound to possible values of `?p`. These 207 subqueries are indeed membership queries, because they check the presence of a concrete triple without variables, e.g., "`dbp:Adam_Thomas rdf:type dbo:Artist`". All values of `?p` that result in a match are solution mappings to the query. This process leads to an evaluation tree, as shown in Figure 4.7.

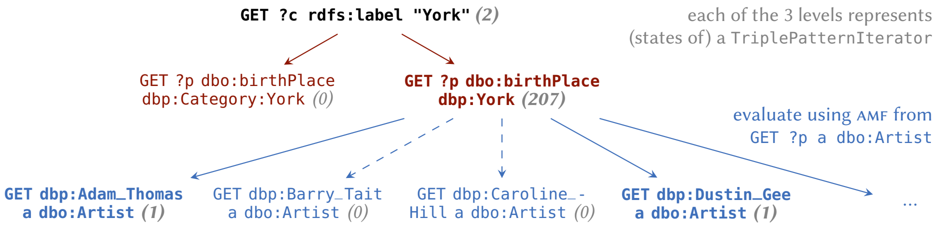

GET ?c rdfs:label "York" (2)
each of the 3 levels represents (states of) a TriplePatternIterator
GET ?p dbo:birthPlace dbp:Category:York (0)
GET ?p dbo:birthPlace dbp:York (207)
evaluate using AMF from GET ?p a dbo:Artist
GET dbp:Adam_Thomas a dbo:Artist (1)
GET dbp:Barry_Tait a dbo:Artist (0)
GET dbp:Caroline_-Hill a dbo:Artist (0)
GET dbp:Dustin_Gee a dbo:Artist (1)
…

Figure 4.7: The triple patterns of Listing 4.8 with the least number of matches at each stage become nodes in the evaluation tree. Note how the third level of consists entirely of membership subqueries (single triples), and can thus be evaluated with the help of an AMF.

As described in Listing 4.3, the whole of Figure 4.7 is executed by a `GraphPatternIterator`, which chains together `TriplePatternIterators` for each of the three levels in the tree. Each `TriplePatternIterator` reads solution mappings from the iterator above it and tries to extend them with mappings for a given triple pattern. For instance, the iterator at level 2 with pattern "`?p dbo:birthPlace ?c`" receives mappings for `?c` from the iterator at level 1. For each `?c`, it tries to find mappings for `?p`, which are then passed on to level 3. Finally, the `TriplePatternIterator` on level 3 with pattern "`?p rdf:type dbo:Artist`" either confirms or rejects mappings depending on whether the triple for a given `?p` exists. This produces a total of 207 requests, which amount to 98% of the total HTTP traffic.

Listing 4.9 presents an extension of the original `TriplePatternIterator` to make use of AMF metadata. When a `TriplePatternIterator` is initiated, the first page of the corresponding TPF for its initial triple pattern is requested (line 2). This fragment typically already resides in the client cache, since it was formerly requested by a `GraphPatternIterator` for count metadata. If the response contains AMF metadata, a *membership test function* is created and assigned to the iterator (line 4). In our example, this translates to a request for the TPF for "`?p rdf:type dbo:Artist`", which contains an AMF for all mappings of `?p`. If no AMF metadata is found, we assign a constant function $True$ that always returns `true` (possible match), so that a verification request is always necessary.

When `GetNext` is called, the `TriplePatternIterator` first reads an upstream mapping μ_s from its source iterator I_s (line 14). Then, we test whether the

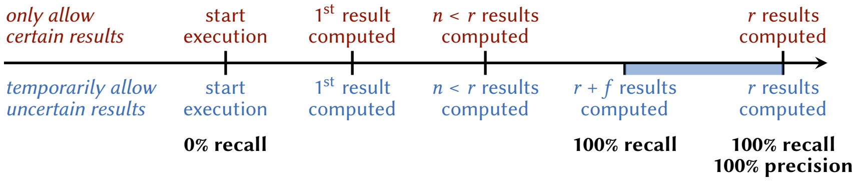


Figure 4.8: This SPARQL query execution timeline compares regular and opportunistic query execution, assuming r total query results and f false positives. Note how both approaches achieve 100% recall and precision at a shared point in the end, but there exists a period during which only opportunistic execution reaches 100% recall (shaded).

triple (pattern) tp' resulting from this mapping is present in the current AMF. If the test returns `true`, we have a true positive or false positive, so the TPF corresponding to tp' is fetched and assigned to the iterator. For instance, if the mapping $\{?p =$ `Adam_Thomas`$\}$ returns `true`, we retrieve the TPF for "`dbp:Adam_-Thomas rdf:type dbo:Artist`" to verify whether this triple is a true or false positive. If the test returns `false`, tp' is a true negative and need not be checked. For instance, if the mapping $\{?p =$ `Barry_Tait`$\}$ returns `false`, we are sure the corresponding TPF is empty, so we do not need to perform the HTTP request.

For each negative AMF result, this proposed extension of the algorithm saves an HTTP request. Depending on the type of query, cumulative savings can be extensive, as with the query in Listing 4.8. The positive results, however, still need to be verified in case false positives would have occurred.

While we cannot eliminate the verification HTTP calls without endangering the correctness (precision) of query results, it is possible to further reduce the query time, as we will discuss in the next section.

4.4.4 Opportunistic query results

Regardless of whether the query execution produces results, the engine spends time on ensuring the result set is valid; it could be that possible result candidates need to be ruled out before being able to decide that the result set is in fact complete. Due to the approximate nature of AMFs, it is possible that at a certain point during Listing 4.9, the in-memory result set R already contains all r valid results. However, they cannot be returned yet, because R can still contain a number of false positives f. Only after the membership of all positive results of the AMF has been verified against the TPF interface, the f false positives can be discarded and all r matches can be returned safely.

For some use cases, it might be acceptable to *temporarily* consider incorrect results, especially if we are able to indicate which results can be trusted and which results cannot. If at first, we optimistically assume that all positive

```
Function TriplePatternIterator.Init()
    Data: A source iterator $I_s$; a triple pattern $tp$; a control $c$ of a $c$-specific
          TPF collection $F$
    $\phi_1 = \langle u_1, u, s, \Gamma_1, M_1, C_1 \rangle \longleftarrow$ GET $c(\{tp\})$, resulting in page 1 of the TPF for $tp$;
    if $\langle$AMF, rdf:type, ms:ApproximateMembershipFunction$\rangle \in M_1$ then
        self.membership_test $\longleftarrow$ AMF where
          $\langle$AMF, rdf:type, ms:ApproximateMembershipFunction$\rangle \in M_1$;
    else
        self.membership_test $\longleftarrow$ $True$ where $\forall x : True(x) =$ true;
    end
    self.current_fragment $\longleftarrow \emptyset$;
end
Function TriplePatternIterator.GetNext()
    Output: The next mapping $\mu_n$ or nil when no such mappings are left
    $\mu \longleftarrow$ nil;
    while $\mu =$ nil do
        while self.current_fragment is empty or only contains read triples do
            self.$\mu_s \longleftarrow I_s$.GetNext();
            return nil if self.$\mu_s =$ nil;
            $tp' \longleftarrow$ self.$\mu_s[tp]$;
            if self.membership_test($tp'$) = true then
                self.current_fragment $\longleftarrow$ GET TPF for $tp'$;
            end
        end
        $t \longleftarrow$ an unread triple from self.current_fragment;
        $\mu \longleftarrow$ a solution mapping $\mu'$ with $\mathrm{dom}(\mu') = \mathrm{vars}(tp)$ and $\mu'[tp] = t$;
    end
    return $\mu \cup$ self.$\mu_s$;
end
```

Listing 4.9: A TriplePatternIterator with support for AMF metadata. To simplify the pseudo-code, the paging details from Listing 4.3 have been omitted from the GetNext() function.

matches of the AMF are actual matches (i.e., we disregard the false positive rate), the client is able to reach 100% recall earlier, temporarily tolerating a precision below 100%. For each of those approximate matches, the client can express the probability that it is valid, namely $1-p$ with p the false-positive rate of the AMF. As membership subqueries progress, the client can update the probability for true positives from $1 - p$ to 1, and retract false positives by setting their probability to 0. This opportunistic method of providing query results is important if fast results and eventual full precision are preferred over slower results with immediate precision. At no point in time, incorrect query results are presented as correct results of the query.

Figure 4.8 compares regular querying and opportunistic querying. Note in particular how both approaches eventually reach 100% recall and precision *at the same time*. In other words, even though the opportunistic algorithm temporarily allows uncertain results and thus a precision of less than 100%, the application eventually obtains the accurate result set. Also, the application that receives the result knows at each moment in time whether a result is certain or not, and can thus decide to either use it or not.

As an example, consider an application that displays photos of artists based on the results of a certain SPARQL query. After a few HTTP calls, the query client returns 50 matches, all of which have a probability of 99%. The application can decide to already start downloading photos of the 50 matching artists, without displaying them to the user yet. Once 48 of the 50 matches are confirmed, the 48 photos can be displayed immediately; only 2 photos need to be discarded. The user thus sees the photos faster than if they had only been retrieved after full precision was achieved. This example indicates that opportunistic query answering has direct concrete uses in Web applications.

4.5 Evaluating Approximate Membership metadata

In the following, we discuss our evaluation of executing SPARQL queries against TPF interfaces with an AMF feature. From these experiments, we aim to assess whether AMFs are a valuable asset in the metadata dimension. Given the presence of AMFs, the client should be able to omit a portion of requests over HTTP, which has a direct impact on the overall execution time. We do not expect much extra load on the server, since an AMF using a non-cryptographic hash function can be computed fast.

4.5.1 Experimental setup

We extended the existing implementations of the TPF client[7] and server[8] to support both Bloom filters and Golomb-coded sets. The server is configured by specifying the AMF and the desired false positive probability. We chose the 32-bit MurMurHash3 hash function for GCS and FNV-1 for the Bloom filter for their ease of implementation. (they perform equivalently in this context). The server calculates a membership function on the fly for each request for a triple pattern with a single variable.

We ran the experiments with different false positive probabilities p: $1/1024 \approx 0.1\%$, $1/128 \approx 1\%$, and $1/64 \approx 1.6\%$. In each experiment, we executed 250 queries generated from 125 diverse WatDiv SPARQL templates on three interfaces: *(a)* regular TPF interface, *(b)* TPF with Bloom filters, and *(c)* TPF with GCS. All three cases were tested with both the original and the optimized client by Van Herwegen et al. [23]; the last two setups were tested with and without opportunistic querying. All experiments were run on a single Amazon EC2 machine

7. https://github.com/LinkedDataFragments/Client.js/tree/amq
8. https://github.com/LinkedDataFragments/Server.js/tree/amq

with an 8-core Intel Xeon E2680 v2 CPU and 15GB DDR3 RAM, using a query timeout of 3 minutes and the WatDiv 100M triples dataset from Aluç et al. [35]. The HTTP requests were routed through an NGINX cache instance to enable HTTP caching and to enforce a realistic Web bandwidth of 1Mbps per request. We published the full result logs online[9].

metric	**# requests**			**1st result time (s)**			**100% recall time (s)**			**total time (s)**		
queries in group	equal	lower	higher	equal	lower	higher	equal	lower	higher	equal	lower	higher
p=1/1024	81 qrs.	126 qrs.	43 qrs.	177 qrs.	0 qrs.	73 qrs.	152 qrs.	3 qrs.	95 qrs.	154 qrs.	1 qry.	95 qrs.
orig. group avg.	*2,953*	*45,213*	*24,312*	*1*	*–*	*7*	*96*	*134*	*67*	*96*	*42*	*67*
avg. difference		*−15,217*	*+10*		*–*	*+6*		*−41*	*+23*		*−32*	*+22*
p=1/128	79 qrs.	134 qrs.	37 qrs.	173 qrs.	0 qrs.	77 qrs.	150 qrs.	3 qrs.	97 qrs.	153 qrs.	1 qry.	96 qrs.
orig. group avg.	*1,469*	*44,712*	*23,623*	*0*	*–*	*7*	*97*	*134*	*66*	*98*	*42*	*66*
avg. difference		*−14,210*	*+5*		*–*	*+5*		*−28*	*+24*		*−32*	*+23*
p=1/64	80 qrs.	129 qrs.	41 qrs.	174 qrs.	0 qrs.	76 qrs.	152 qrs.	3 qrs.	95 qrs.	156 qrs.	1 qry.	93 qrs.
orig. group avg.	*2,340*	*44,842*	*24,626*	*1*	*–*	*7*	*96*	*134*	*66*	*97*	*42*	*66*
avg. difference		*−13,341*	*+15*		*–*	*+4*		*−41*	*+21*		*−33*	*+21*

Table 4.3: Comparison of regular TPF versus TPF with Bloom filter setup (greedy TPF algorithm)

metric	**# requests**			**1st result time (s)**			**100% recall time (s)**			**total time (s)**		
queries in group	equal	lower	higher	equal	lower	higher	equal	lower	higher	equal	lower	higher
p=1/1024	83 qrs.	123 qrs.	44 qrs.	195 qrs.	0 qrs.	55 qrs.	160 qrs.	0 qrs.	90 qrs.	167 qrs.	0 qrs.	83 qrs.
orig. group avg.	*2,271*	*45,598*	*26,919*	*1*	*–*	*10*	*94*	*–*	*70*	*91*	*–*	*72*
avg. difference		*−11,761*	*+18*		*–*	*+8*		*–*	*+15*		*–*	*+16*
p=1/128	83 qrs.	132 qrs.	35 qrs.	196 qrs.	0 qrs.	54 qrs.	154 qrs.	0 qrs.	96 qrs.	153 qrs.	0 qrs.	97 qrs.
orig. group avg.	*2,152*	*45,924*	*21,168*	*1*	*–*	*11*	*96*	*–*	*67*	*98*	*–*	*66*
avg. difference		*−11,594*	*+5*		*–*	*+8*		*–*	*+16*		*–*	*+16*
p=1/64	81 qrs.	122 qrs.	47 qrs.	199 qrs.	0 qrs.	51 qrs.	167 qrs.	2 qrs.	81 qrs.	164 qrs.	2 qrs.	84 qrs.
orig. group avg.	*2,930*	*45,032*	*26,602*	*1*	*–*	*11*	*91*	*122*	*72*	*93*	*122*	*70*
avg. difference		*−10,521*	*+31*		*–*	*+7*		*−3*	*+13*		*−3*	*+12*

Table 4.4: Comparison of regular TPF versus TPF with GCS setup (greedy TPF algorithm)

4.5.2 Impact on the number of HTTP requests

Tables 4.3 to 4.6 summarize the results of the experiments. They compare each AMF-enabled setup against a regular TPF client/server setup, grouping each of the 250 queries on whether they resulted in an *equal*, *lower*, or *higher* measurement for *(a)* number of requests, *(b)* time to first result, *(c)* time to 100%

9. https://github.com/LinkedDataFragments/TPF-Membership-Metadata-Results

metric	**# requests**			**1st result time (s)**			**100% recall time (s)**			**total time (s)**		
queries in group	equal	lower	higher	equal	lower	higher	equal	lower	higher	equal	lower	higher
p=1/1024	82 qrs.	155 qrs.	13 qrs.	166 qrs.	0 qrs.	84 qrs.	200 qrs.	0 qrs.	50 qrs.	173 qrs.	0 qrs.	77 qrs.
orig. group avg.	*1,590*	*18,240*	*11,387*	*1*	–	*5*	*120*	–	*71*	*110*	–	*69*
avg. difference		*−4,920*	*+2*		–	*+5*		–	*+21*		–	*+18*

Table 4.5: Comparison of regular TPF versus TPF with Bloom filter setup (optimized TPF algorithm)

metric	**# requests**			**1st result time (s)**			**100% recall time (s)**			**total time (s)**		
queries in group	equal	lower	higher	equal	lower	higher	equal	lower	higher	equal	lower	higher
p=1/1024	87 qrs.	147 qrs.	16 qrs.	199 qrs.	0 qrs.	51 qrs.	203 qrs.	0 qrs.	47 qrs.	209 qrs.	0 qrs.	41 qrs.
orig. group avg.	*2,743*	*18,326*	*10,816*	*1*	–	*9*	*120*	–	*74*	*114*	*– 88*	
avg. difference		*−1,154*	*+3*		–	*+5*		–	*+14*		–	*+11*

Table 4.6: Comparison of regular TPF versus TPF with GCS setup (optimized TPF algorithm)

recall (i.e., with opportunistic querying enabled), and *(d)* total query execution time. The number of queries per group is indicated, together with their average measurement value in the regular setup, and the average decrease or increase in respectively the *lower* and *higher* groups. For example, the top-left value cell of Table 4.3 shows that, for Bloom filters with $p = 1/1024$, 126 queries had a lower number of HTTP requests; for each of these 126 queries, the regular setup needed on average 45,213 requests, whereas the AMF-enabled setup required 15,217 fewer requests.

Our experiments show that, with $p = 1/1024$, AMF metadata decreases the number of HTTP calls for roughly half of all considered queries (Bloom: 126 queries or 50.4%; GCS: 123 queries or 49.2%). As expected from the analysis in Section 4.4, those queries that benefit from improvements are queries with relatively many HTTP requests: the average number of requests per query in the *lower* group is 45,213 (GCS: 45,598), compared to 2,953 (GCS: 2,271) for queries that do not improve. The improvements let us conclude that a substantial number of these 45,000+ requests per query were membership subqueries; the AMF-based query algorithm manages to decrease their number by 15,217 (GCS: 11,761) on average. 43 queries (GCS: 43) result in a slightly higher number of requests, albeit negligible compared to the total number: 10 versus 24,312 (GCS: 18 / 26,919). Note that in general, the number of requests per query is very high because of the potentially high number of results in the WatDiv dataset. While numbers of this scale clearly highlight query patterns, many real-world queries can be evaluated with tighter constraints.

A similar pattern arises with the optimized TPF algorithm [23], which consumes fewer HTTP requests overall because of full client-side joins, but has potentially longer query times for the same reason. Even more queries benefit from lower request numbers: 155 (62%) for Bloom and 147 (58.8%) for GCS. We see a reduction of roughly the same ratio, both with Bloom filters and GCS, although the absolute request numbers are lower.

The observations generalize to the cases for $p = 1/128$ and $p = 1/64$, albeit with slightly different observations. As is expected from a higher number of false positives, we see a decreasing average gain with increasing p. Interestingly, we see the number of queries with fewer HTTP requests increase slightly for higher p values; we assume this is correlated with the smaller response size, which allows for a higher throughput.

4.5.3 Impact on query execution time

In all cases (excluding 1 or 2 exceptions), both the first result times and total query times remain the same or even increase, contrarily to what we had expected. As Tables 4.3 and 4.4 indicate, about one in three queries have their execution time prolonged with about 20 seconds, or a third of their time. This prolongation is higher for Bloom filters than GCS, which see a more limited effect absolutely (18 seconds) and proportionally (about a quarter). The cause of these elevated query times is likely the increased response size: since the server automatically sends AMFs for all patterns with one variable (even if the client does not use the AMF), the server-side computation time and client-side retrieval time increase. Given a connection of 1Mbps and on-the-fly AMF generation, as in this experiment, the decreased number of requests is apparently insufficient for the considered queries and dataset to result in temporal gains. This is confirmed by the fact that GCS performs better, as GCS representations are encoded more efficiently.

Interestingly, higher false-positive probabilities do not have a profound effect on query time. For the given constraints, the higher number of requests seems to be compensated by the decreased complexity of generating, transferring, and interpreting AMFs. This is an indication that further experimentation with low probabilities might be beneficial.

The prolonged total query time also hinders the effectiveness of opportunistic querying. Whereas its goal is to achieve full recall earlier—at the expense of temporarily allowing <100% precision—the slower overall execution prevented a globally positive result. The potential benefit of opportunistic querying is evidenced by the 3 queries that, with Bloom filters, achieve 100% recall 41 seconds—about a third—earlier. Since opportunistic results have no negative influence on query time, the increased recall times for ±95 queries must be entirely due to the slower speed of the AMF approach under the 1Mbps and on-the-fly constraints. Should we succeed in speeding up AMF generation and/or transfer time, we could expect to see a broader influence of opportunistic results. Furthermore, the number of false positives that needed to be revoked was either 0 or 1 for all of the considered queries, revealing a low temporary impact on precision.

Further research will need to assess the relation of this observation to on-the-fly generation and bandwidth, and perhaps also even higher false positive rates.

4.5.4 Impact on server load

Finally, we measured the average CPU load during query execution for two different AMF configurations and two different false positive probabilities. Compared to the normal server CPU usage (9.2%), the AMF configurations show an increase of 1.6% ($p = 1/1024$), 2% ($p = 1/128$) and 5.7% ($p = 1/64$) for Bloom, and 1% ($p = 1/1024$), 1.6% ($p = 1/128$), and 1.9% ($p = 1/64$) for GCS. This is a very acceptable overhead which does not impact the server's low-cost nature. Bloom has a higher impact than GCS because of the many hashes it needs to calculate, which apparently outweigh the overhead of Golomb compression. Note that all AMF metadata is created at query time and can still benefit from pre-computation and/or caching.

4.6 Conclusion

Application development for the Web of Linked Data can be stimulated by lowering server cost for Web APIs to RDF. The resulting cheaper infrastructure enables more organizations to host queryable Linked Data and at the same time, ensure a more reliable service through more uptime (Research Question 1). Triple Pattern Fragments (TPF) therefore restricts the granularity of the interface to `?s ?p ?o` patterns and provides clients with *cardinality* metadata. Hence, TPF introduces a new trade-off mix for client-side SPARQL execution with *(a)* low server cost; *(b)* higher execution time; and *(c)* higher bandwidth usage.

Results show that, with increasing client numbers, the TPF interface generally has low and more constant CPU load than SPARQL endpoints during query execution. Querying also benefits strongly from regular HTTP caching. As expected, response times are significantly higher, not receiving all results within the timeout window for some queries. However, they are more stable than with SPARQL endpoints and, due to incremental results, cause the first result to drop in before the timeout. These findings generalize towards real-world datasets such as DBpedia. A vast majority of queries stays well below the four-second limit, despite being affected by the dataset size. Also, we detected a strong influence of the query type, especially when non-BGP SPARQL constructs are involved. Note that these results were obtained with *existing* SPARQL benchmarks that focus on performance, not server cost, which make these results even more promising. Overall we can validate Hypothesis 1.

In general, response times correlate with the number of requests; hence, more requests result in slower query response times, aggravated by the network delay (Research Question 2). More compact response formats do decrease the overall overhead, but compressing the reponses by GZIP (commonly used within the HTTP protocol) achieve the same benefits. For typical page sizes (e.g., the default 100) of a TPF interface, serialization, and deserialization cost even increase to the time per request. Yet, a re-evalution is desired with alternative query algorithms and the presence of HTTP/2. For certain types of queries, most of that request overhead are in fact membership subqueries. However, adding additional Approximate Membership Functions (AMF) metadata to the TPF interface does not show immediate improvement. AMFs do drastically reduce the number of requests for half of the tested queries, with some queries experiencing little overhead thanks to local caching. Hence, these findings validate Hypothesis 2. Because of long delays introduced to generate AMFs, though, the total execution time is in fact higher on average.

Although it does not affect the low-cost nature of the server—CPU load increase is only limited, real-time computation of AMF metadata should be avoided. We recommend to pre-compute or pre-cache it in advance. Also, serving AMFs in a separate resource, and link to it in the response, could reduce the transfer and generation overhead. An intelligent client can then decide when to download and use membership metadata, for example based on the query type. This avoids a computational overhead for queries that are not improved. Enabling opportunistic querying can already compensate this: retracting results after validation is rare and only effects a small number of results. A separate AMF resource can further exploit the HTTP caching benefits of TPF. While Bloom filters are preferred for lower computation time, the smaller size of Golomb-coded sets would prevail in the presence of caching.

With respect to dereferencing, TPF interfaces mitigate the authority issue that occurs with Linked Data documents; their hypermedia form allows clients to inquire about any URI, regardless of whether it resides in the server's dataspace. Therefore, we can discover information about a given subject in different interfaces. In fact, TPF can be considered as additional constraints to the Linked Data principles [55]: each TPF with a fixed subject (`{ <s> ?p ?o }`) has its own HTTP URI, represents triples about a certain subject, and includes links to other documents that allow to discover more things (all related TPFs). At the same time, the interface remains fully compatible with dereferencing. For instance, dereferencing the URL http://dbpedia.org/resource/Walt_Disney could result in an HTTP 303 redirect to the fragment resource http://fragments.dbpedia.org/en?subject=http%3A%2F%2Fdbpedia.org%2Fresource%2FWalt_Disney, which contains all triples with this particular URL as a subject. Moreover, TPF servers can present their *own* (not just DBpedia's) metadata of Walt Disney; and all resources that have Walt Disney as object.

Finally, the transparent and self-descriptive nature of TPF responses has its merits. Clients can find out dynamically if this and other features are supported. This was illustrated by adding AMF metadata to the existing TPF interface. The server can choose freely whether or not to add metadata to a certain response; clients can reactively use metadata when possible, or ignore it when they do not support or need it.

This chapter was partly based on the publications:

Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. "Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web." In: *Journal of Web Semantics* 37–38 (Mar. 2016), pp. 184–206. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.03.003. URL: http://linkeddatafragments.org/publications/jws2016.pdf

Miel Vander Sande, Ruben Verborgh, Joachim Van Herwegen, Erik Mannens, and Rik Van de Walle. "Opportunistic Linked Data Querying through Approximate Membership Metadata." In: *The Semantic Web – ISWC 2015.* Ed. by Marcelo Arenas et al. Vol. 9366. Lecture Notes in Computer Science. Bethlehem, PA: Springer International Publishing, Oct. 2015, pp. 92–110. ISBN: 978-3-319-25006-9. DOI: 10.1007/978-3-319-25007-6_6. URL: http://linkeddatafragments.org/publications/iswc2015-amf.pdf

References

[1] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. "SPARQLWeb-Querying Infrastructure: Ready for Action?" In: *The 12th International Semantic Web Conference.* Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz. Nov. 2013.

[2] Olaf Hartig. "An Overview on Execution Strategies for Linked Data Queries." In: *Datenbank-Spektrum* 13.2 (2013), pp. 89–99.

[3] Yannis E Ioannidis. "Query optimization." In: *ACM Computing Surveys (CSUR)* 28.1 (1996), pp. 121–123.

[4] Yannis Ioannidis. "The History of Histograms (Abridged)." In: *The 29th International Conference on Very Large Data Bases.* Vol. 29. VLDB '03. Berlin, Germany: VLDB Endowment, 2003, pp. 19–30. ISBN: 0-12-722442-4. URL: http://dl.acm.org/citation.cfm?id=1315451.1315455.

[5] Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Bottom-up Web APIs with self-descriptive responses." In: *The 6th International Workshop on Modeling Social Media.* Feb. 2015.

[6] Roy Thomas Fielding. *REST APIs must be hypertext-driven.* Oct. 2008. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[7] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. "Querying Datasets on the Web with High Availability." In: *The 13th International Semantic Web Conference.* Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris

Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, Oct. 2014, pp. 180–196.

[8] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-query/.

[9] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. "Binary RDF Representation for Publication and Exchange (HDT)." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (Mar. 2013), pp. 22–41.

[10] Luca Matteis. *Restpark: Minimal RESTful API for Retrieving RDF Triples*. 2013. URL: http://lmatteis.github.io/restpark/restpark.pdf.

[11] Ruben Verborgh. *Triple Pattern Fragments*. Unofficial Draft. Hydra W3C Community Group, Sept. 29, 2017. URL: http://www.hydra-cg.com/spec/latest/triple-pattern-fragments/.

[12] Gavin Carothers. *RDF 1.1 N-Quads*. Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/n-quads/.

[13] Chris Bizer and Richard Cyganiak. *RDF 1.1 TriG*. Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/trig/.

[14] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. *JSON-LD 1.0*. Recommendation. World Wide Web Consortium, Jan. 16, 2014. URL: http://www.w3.org/TR/json-ld/.

[15] Markus Lanthaler and Christian Gütl. "Hydra: A Vocabulary for Hypermedia-Driven Web APIs." In: *The 6th Workshop on Linked Data on the Web*. Vol. 996. May 2013.

[16] Mike Amundsen. "Hypermedia Types." In: *REST: From Research to Practice*. Ed. by Erik Wilde and Cesare Pautasso. Springer, 2011, pp. 93–116.

[17] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL." In: *ACM Transactions on Database Systems* 34.3 (Sept. 2009), 16:1–16:45. ISSN: 0362-5915.

[18] Pablo Barceló. "Querying Graph Databases." In: *The 32nd Symposium on Principles of Database Systems (PODS)*. 2013.

[19] Olaf Hartig. "SPARQL for a Web of Linked Data: Semantics and Computability." In: *The Semantic Web: Research and Applications*. Ed. by Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti. Springer, 2012, pp. 8–23.

[20] Olaf Hartig. "How Caching Improves Efficiency and Result Completeness for Querying Linked Data." In: *The 4th Workshop on Linked Data on the Web*. Ed. by Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas. Mar. 2011.

[21] Goetz Graefe. "Query Evaluation Techniques for Large Databases." In: *ACM Computing Surveys (CSUR)* 25.2 (June 1993), pp. 73–169. ISSN: 0360-0300.

[22] Joseph M. Hellerstein, Michael Stonebraker, and James Hamilton. "Architecture of a Database System." In: *Foundations and Trends in Databases* 1.2 (2007), pp. 141–259. DOI: 10.1561/1900000002.

[23] Joachim Van Herwegen, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Query Execution Optimization for Clients of Triple Pattern Fragments." In: *The 12th Extended Semantic Web Conference*. Ed. by Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann. June 2015.

[24] Maribel Acosta and Maria-Esther Vidal. "Networks of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data." In: *The Semantic Web – ISWC 2015*. Ed. by Marcelo Arenas et al. Vol. 9366. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 111–127. ISBN: 978-3-319-25006-9.

[25] Orri Erling and Ivan Mikhailov. "Virtuoso: RDF Support in a Native RDBMS." In: *Semantic Web Information Management*. Ed. by Roberto de Virgilio, Fausto Giunchiglia, and Letizia Tanca. Springer, 2010, pp. 501–519. ISBN: 978-3-642-04328-4.

[26] Michael Grobe. "RDF, Jena, SPARQL and the Semantic Web." In: *The 37th Annual ACM SIGUCCS Fall Conference: Communication and Collaboration*. 2009. ISBN: 978-1-60558-477-5.

[27] Christian Bizer and Andreas Schultz. "The Berlin SPARQL benchmark." In: *International Journal on Semantic Web and Information Systems* 5.2 (2009), pp. 1–24.

[28] Fiona Fui-Hoon Nah. "A study on tolerable waiting time: how long are web users willing to wait?" In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163.

[29] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. "DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data." In: *The 9th International Semantic Web Conference*. 2011. ISBN: 978-3-642-25072-9.

[30] David Beckett. *RDF 1.1 N-Triples*. Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/n-triples/.

[31] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. *RDF 1.1 Turtle*. Recommendation. World Wide Web Consortium, Feb. 25, 2014. URL: http://www.w3.org/TR/turtle/.

[32] Andy Seaborne. *RDF Binary using Apache Thrift*. Ed. by Andy Seaborne. Sept. 29, 2017. URL: http://afs.github.io/rdf-thrift.

[33] Jeen Broekstra. *Binary RDF in Sesame*. Nov. 2011. URL: http://www.rivuli-development.com/2011/11/binary-rdf-in-sesame.

[34] Javier D Fernández, Alejandro Llaves, and Oscar Corcho. "Efficient RDF Interchange (ERI) Format for RDF Data Streams." In: *The 13th International Semantic Web Conference*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Springer, 2014, pp. 244–259.

[35] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. "Diversified stress testing of RDF data management systems." In: *The 13th International Semantic Web Conference*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Springer, 2014, pp. 197–212.

[36] Annita N. Wilschut and Peter M. G. Apers. "Dataflow Query Execution in a Parallel Main-memory Environment." In: *The 1st International Conference on Parallel and Distributed Information Systems*. PDIS '91. Miami, Florida, USA: IEEE Computer Society Press, 1991, pp. 68–77. ISBN: 0-8186-2295-4. URL: http://dl.acm.org/citation.cfm?id=382009.383658.

[37] Mike Belshe, Martin Thomson, and Roberto Peon. *Hypertext Transfer Protocol version 2 (HTTP/2)*. Request For Comments 7540. Internet Engineering Task Force, 2015. URL: https://tools.ietf.org/html/rfc7540.

[38] Elias Szabo-Wexler. *Approximate Membership of Sets: A Survey*. Survey. Carnegie Mellon University, Jan. 2014. URL: http://www.cs.cmu.edu/~lblum/flac/Presentations/Szabo-Wexler_ApproximateSetMembership.pdf.

[39] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors." In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692.

[40] Felix Putze, Peter Sanders, and Johannes Singler. "Cache-, hash-, and space-efficient Bloom filters." In: *Journal of Experimental Algorithmics* 14 (2009), p. 4.

[41] Michael Mitzenmacher. "Compressed Bloom filters." In: *Transactions on Networking* 10.5 (2002).

[42] R.G. Gallager and David C. Van Voorhis. "Optimal source codes for geometrically distributed integer alphabets." In: *Transactions on Information Theory* 21.2 (Mar. 1975), pp. 228–230. ISSN: 0018-9448. DOI: 10.1109/TIT.1975.1055357.

[43] Imen Filali, Francesco Bongiovanni, Fabrice Huet, and Françoise Baude. "A survey of structured P2P systems for RDF data storage and retrieval." In: *Trans. large-scale data-and knowledge-centered systems*. 2011.

[44] Juan Li and Son Vuong. "OntSum: A Semantic Query Routing Scheme in P2P Networks Based on Concise Ontology Indexing." In: *21st International Conference on Advanced Information Networking and Applications (AINA'07)*. IEEE, May 2007, pp. 94–101. DOI: 10.1109/AINA.2007.104.

[45] Padmashree Ravindra, Seokyong Hong, HyeongSik Kim, and Kemafor Anyanwu. "Efficient Processing of RDF Graph Pattern Matching on MapReduce Platforms." In: *The 2nd International Workshop on Data Intensive Computing in the Clouds.* Seattle, Washington, USA: ACM, 2011, pp. 13–20. ISBN: 978-1-4503-1144-1. DOI: 10.1145/2087522.2087527.

[46] Felix Heine. "Scalable P2P based RDF querying." In: *The 1st international conference on Scalable information systems.* 2006.

[47] Xiaofei Zhang, Lei Chen, and Min Wang. "Towards efficient join processing over large RDF graphs using MapReduce." In: *Scientific and Statistical Database Management.* 2012, pp. 250–259.

[48] Hai Huang and Chengfei Liu. "Estimating Selectivity for Joined RDF Triple Patterns." In: *Conference on Information and Knowledge Management* (2011), pp. 1435–1444. DOI: 10.1145/2063576.2063784.

[49] Thomas Neumann and Gerhard Weikum. "Scalable Join Processing on Very Large RDF Graphs." In: *The International Conference on Management of Data.* ACM, 2009, pp. 627–640. ISBN: 978-1-60558-551-2. DOI: 10.1145/1559845.1559911.

[50] Cosmin Basca and Abraham Bernstein. "Avalanche: Putting the spirit of the Web back into Semantic Web querying." In: *Scalable Semantic Web Knowledge Base Systems.* 2010, pp. 64–79.

[51] Katja Hose and Ralf Schenkel. "Towards Benefit-based RDF Source Selection for SPARQL Queries." In: *The 4th International Workshop on Semantic Web Information Management.* SWIM '12. Scottsdale, Arizona: ACM, 2012, 2:1–2:8. ISBN: 978-1-4503-1446-6.

[52] Eyal Oren, Christophe Guéret, and Stefan Schlobach. "Anytime query answering in RDF through evolutionary algorithms." In: *Lecture Notes in Computer Science* 5318 (2008), pp. 98–113. ISSN: 0302-9743. DOI: 10.1007/978-3-540-88564-1-7.

[53] Xu Pu, Jianyong Wang, Ping Luo, and Min Wang. "AWETO: Efficient incremental update and querying in RDF storage system." In: *The 20th international conference on information and knowledge management.* ACM, 2011, pp. 2445–2448.

[54] Miel Vander Sande and Ruben Verborgh. *LDF Membership Metadata.* Unofficial Draft. Hydra W3C Community Group, 2013. URL: http://www.hydra-cg.com/spec/latest/linked-data-fragments/membership-metadata/.

[55] Tim Berners-Lee. *Linked Data – Design issues.* Ed. by Tim Berners-Lee. July 27, 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html (visited on 06/18/2009).

Shut up, brain. I got friends now. I don't need you anymore.

— *Lisa Simpson*

Federation of interfaces 5

A Web of Linked Data consists of multiple data sources published through multiple interfaces. Hence, virtually integrating datasets through querying a federation of interfaces has been given much attention in research. However, the high costs of available live queryable Linked Datasets have not yet made this deployable in practice. Many institutions instead depend on an aggregator to offer query access over physically integrated datasets. This chapter argues that virtual integration is a better fit for many data publishers with the right infrastructure, and therefore investigates Research Question 3. To enable query execution over a federation of Triple Pattern Fragments interfaces, we introduce and evaluate a mediator layer that extends current TPF clients (Hypothesis 3). Existing source selection techniques—determining relevant sources to evaluate a query—are compatible with this approach and recommended to be used in conjunction. Hence, we also provide an overview of the most prominent approaches from literature.

The Web is a fully distributed system—and thus so is the Web of Linked Data. Within this enormous collection, each data source specializes in its very own part of the truth. Some of them, like DBpedia [1], contain essential facts about a broad range of subjects; others, like Linked Open Drug Data [2], offer a comprehensive corpus of triples about highly select topics. This knowledge only reaches its highest potential when we are able to answer any non-trivial query *across* different data sources. As a result, the need for such *federated queries* intensifies as the Linked Open Data cloud is trending toward a more decentralized graph structure, with additional linking hubs besides DBpedia arising [3]. Federation is thus necessary to achieve the Web of Data vision [4]: a global, machine-understandable dataspace with web-scale integration and interoperability.

Unfortunately, we do not see federated querying over distributed interfaces in practice yet. One reason is that they suffer most from the low availability of live queryable Linked Datasets. The joint availability of all interfaces is calculated by the product of their individual availability, thus it can only decrease with the number of sources. Besides that, there is an interoperability issue where some datasets are published as a SPARQL endpoint and some are not. Most data publishers in the LAM or Open Data community make their collections of RDF statements available as Linked Datasets for batch download. Then, one or more aggregators step in and collect the distributed datasets and publish a merged dataset either—again—for batch download or as a machine-queryable endpoint.

This *physical integration* approach is cost-effective for institutions that expose Linked Datasets and aggregators often add value, for example, by performing data cleansing and mapping equivalent URIs. But when the aggregated dataset is exposed as a centralized query endpoint, it cannot really be considered *federated* querying. Furthermore, besides the discussed infrastructure scalability issues, the approach also has a some important drawbacks.

First, data in different organizations evolves at a different pace. Keeping an aggregated dataset continuously synchronized with the evolving distributed datasets is a non-trivial technical challenge [5]; tackling it in a realistic manner would necessarily involve additional infrastructure (and hence investment) at the end of the institutions that expose them. Lacking this, at any moment in time, it is uncertain whether or not an aggregated dataset is in sync with the state of the datasets it merges.

Second, data publishers often fear loosened control when making their Linked Datasets available for reuse. Considering their own datasets as highly curated, they might be reluctant to allow a merge with datasets perceived to be of lower quality—especially in the LAM community. In addition, descriptions originating from different institutions are commonly made available under different licenses, making it difficult to understand what the terms of use for aggregated descriptions are, especially when licenses are in conflict. Although maintaining the data provenance for the entire aggregation could ensure a basic sense of control, this burdens institutions to share this information as well.

5.1 Reviving Virtual integration with low-cost interfaces

An interesting alternative for institutions is to expose their Linked Datasets through their own query endpoints. In this case, client applications query distributed datasets, benefiting from a uniform query interface. This *virtual integration* approach is more expensive for institutions because it requires maintaining these endpoints. Also, it yields the non-trivial *source selection problem* [6], as client applications need to limit the distributed datasets they consult for any given query in order to balance completeness of results with acceptable latency.

Source selection is an important step in federated SPARQL query execution frameworks [7, 8, 9]. It usually involves pre-computed summaries and/or the retrieval of extra (meta-)data from the endpoints, and happens as a *separate* step before the actual execution. While this step is intended to reduce the number of requests to servers, and hence the overall query evaluation time, source selection itself also takes time. Since source selection can be performed *independently* of the actual query execution, existing source selection algorithms can be used in conjunction with different execution strategies.

Despite its issues, virtual integration does not have the significant drawbacks of physical integration described earlier. If technological advances can be made that ameliorate problems related to data source selection, uniform access for clients, and maintenance costs, it is an attractive alternative to avoid conflicts on policy level. Triple Pattern Fragments is therefore an interesting approach to revive the virtual integration strategy. Due to its low-cost server-side interface, it counters the availability issues of SPARQL endpoints and makes hosting an own query endpoint affordable.

Federated query processing has been studied in the context of SPARQL endpoints, where most SPARQL endpoint federation frameworks apply a client-server architecture. The TPF query setup from Section 4.2 implements this paradigm natively, thereby making an extension toward multiples sources straightforward. Hence, we introduce a TPF-specific federation mediator in Section 5.3. First, however, it is important to be aware of what source selection techniques exists in literature, since such step can optionally precede the proposed mediator.

5.2 Source selection for federated query processing

Source selection determines which interfaces potentially can contribute to the result set of a particular query, and happens when the list of available dataset interfaces is known (for instance through manual or automatic discovery; see Chapter 6). A client evaluates a federated SPARQL query by decomposing it in subqueries that retrieve and join partial results. Therefore, this process is known to be an important performance factor [6]: an overestimation of candidate sources can increase execution time, an underestimation can decrease the recall of results. The exact impact on query performance (e.g., execution time), however, depends on several aspects, such as the number of sources, the amount of empty results, source response time, and parallel execution.

Rakhmawati et al. [10] identified three types of federation frameworks: *(a)* federation over SPARQL endpoints, *(b)* federation using Linked Data traversal, and *(c)* federation over custom repository APIs. The first type expects all data sources to be exposed using SPARQL endpoints. The second type relies on dereferenceable Linked Data documents, exposed by data providers that apply the Linked Data principles; query results are constructed by following links and looking up URIs. In most federation frameworks, source selection happens triple pattern-wise, i.e. sources are selected for each triple pattern in the query, using the four source-selection strategies, which are discussed below, separately or jointly.

ASK queries Schwarte et al. [11] presented FedX, which sends ASK queries to a predefined list of sources to check whether they contain triples matching a given pattern. Despite the many HTTP requests this requires, it is one of the best performing systems in terms of execution time.

Dataset profiling To capture important meta-information about a dataset, compact descriptions (e.g., the Vocabulary of Interlinked Datasets VoID [12] to describe RDF datasets) are automatically generated. This includes statistical properties (e.g., average number of distinct subjects and the co-occurrence frequency of URIs in triples), patterns (e.g., topics, clusters) [13, 14], and content information (e.g., present classes and properties). These dataset profiles are generated by the server, but shipped on-demand to the client. With this information, a source selection process can estimate the so-called *exclusive groups*—sets of triple patterns that only yield results for a specific source—to avoid unnecessary requests.

The extension [15] to the ANAPSID system [7] gathers information about the distinct predicates of data sources and applies smart heuristics to estimate the source selection and improve query planning. Recently, more lightweight summaries are used in HIBISCUS [8]. For each distinct predicate, this approach gathers URI authorities for the subjects and predicates.

Some works have studied scalable methods to profile huge datasets using MapReduce [16, 17], employable on heavy infrastructure such as cloud-systems. Works in this area target a very broad spectrum of applications by generating as much descriptive information as possible. In contrast, our work puts infrastructural constraints on publishing Linked Data servers, so it can sufficiently scale on the Web. Limited load, as well as minimal size of the description are preferred to expressiveness.

Data indexing Before or during query execution, the federation system gathers statistics about sources. In contrast to data profiling, the client, not the server, constructs the metadata with specific queries. One approach is to build a source model on a schema-level. Paret et al. [18] and Li, Niu, and Zhang [19] construct a *Web of Linked Classes* based on classes and their relations, optionally augmented with instance statistics. Umbrich et al. [20] published an extensive report on *data summaries* for live Linked Data querying. Harth et al. [21] construct QTree indexes, which summarize instance- and schema-level elements of a dataset in a hierarchical structure combining R-trees [22] and histograms. Quilitz and Leser [23] presented the early system DARQ, where an index of distinct predicates is first composed to select candidate sources.

Caching Cached data can also increase efficiency for future (partial) re-execution of queries. Although not using data indexing as such, FedX [11] extensively caches prior source selection output for this purpose.

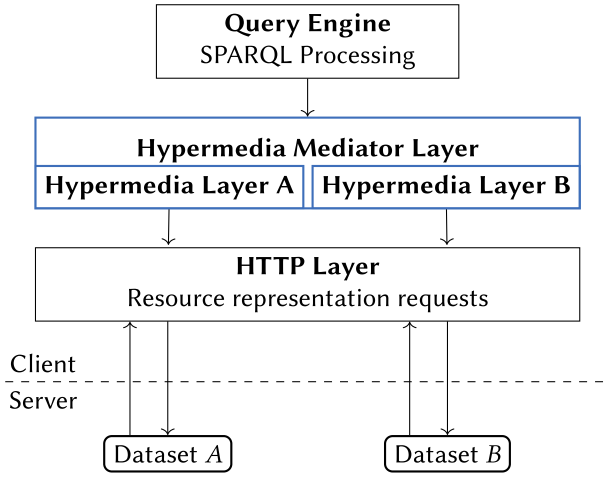


Figure 5.1: A mediator layer adds support for querying a federation of TPF interfaces with an unaltered query engine and HTTP layer.

5.3 Querying a federation of TPF interfaces

As an enhancement to the TPF client introduced in Section 4.2, we propose a *mediator* layer to query multiple TPF interface instances with a single query. A mediator is a software module that abstracts a collection of data resources for a higher software layer [24], making them independent from each other. In this case, each TPF interface acts as a datasource *wrapper*, with the mediator creating a unified TPF view over the interfaces [25]. As a result, the task of a TPF client when executing a (regular) SPARQL query over a federation of TPF interfaces is to compute results identical to those it would obtain when the query would be evaluated over a single TPF interface that combines the data of all considered interfaces. That is, a user can "query using a classical query language against the federated schema with an illusion that he or she is accessing a single system" [26].

Figure 5.1 extends the Hypermedia layer from the original architecture in Figure 4.1. The mediator federates TPF requests to candidate sources and uses run time *source elimination* during query execution to prevent HTTP requests we know would not result in a match. Source elimination either refines the optional source selection step at every iteration by excluding more specific patterns, or, if no prior source selection was performed, acts as a runtime optimization.

After each query, the list of eliminated sources is cleared by default, but it can also be stored to aid subsequent queries. In case sources change, a refresh strategy should be in place to ensure completeness.

The client is given a query, and a pre-defined list of TPF interfaces (instead of a single TPF interface). The mediator creates an abstraction layer to the fragment request operation: all interfaces are exposed to the query algorithm as a single interface. Initially, all interfaces are marked as possible candidates for each triple pattern (unless explicitly ruled out by the optional source selection step). Each time the client requests a TPF, the mediator consults all candidate interfaces for the same triple pattern.

When a TPF interface returns an empty fragment for a certain triple pattern, it becomes an eliminated source for that pattern, which is stored in an interface-specific elimination list. For each triple pattern requested from this interface, we check the elimination list for a possible *ancestor* pattern. A tp_a is an ancestor of tp_b if each term of tp_a is either a variable or equal to the corresponding term of tp_b. If an ancestor pattern is found, the interface is no longer marked as a candidate, since it has no matches for the ancestor pattern, nor for more specific patterns.

For all interfaces, the fragments' data and metadata are merged in a non-blocking way. As if processing a single Triple Pattern Fragment, all triples are read from a single iterator, regardless of when they arrive, or from which source they originate. This iterator is depleted when all sources are probed and the last data triple is consumed. The mediator is therefore non-blocking: the first results can already be processed before the iterator is depleted..

The count metadata are combined into a single value using an aggregation function $\vartheta(tp_j)$, which is a cost function that can be optimized to the type of interface. For example, $\vartheta(tp_j)$ can be a *weighted* sum, taking into account practical differences between N servers, such as response time, page size, etc. In our implementation, we chose an ordinary sum for simplicity; i.e., $\vartheta(tp_j) = \sum_{i=1}^{N} \mathrm{cnt}_{ij}$.

5.4 Experiments

We conclude this chapter by evaluating whether Triple Pattern Fragments can offer an efficient architecture for SPARQL query evaluation over a federation of interfaces. Hence, we aim at results in recall and query execution time similar to state-of-the-art SPARQL query federation frameworks under public network latency.

5.4.1 Experimental setup

We implemented the mediator approach from Section 5.3 in the TPF client (with source elimination, but without source selection), which now accepts a *set* of TPF interfaces and a SPARQL query. As the previously considered benchmarks only use a single knowledge graph, we chose the popular FedBench benchmark [27]. FedBench relies on real-world authoritative datasets that are prominent in the Web of Linked Data and provides a federated query mix. The

dataset	**# triples**	**# distinct subjects**	**# distinct predicates**	**# distinct objects**
DBPedia subset	42,849,609	9,495,865	1,063	13,620,028
NY Times	335,206	21,666	36	191,538
LinkedMDB	6,147,996	694,400	222	2,052,959
Jamendo	1,049,647	335,925	26	440,686
Geonames	107,950,085	7,479,714	26	35,799,392
SW Dog Food	103,465	11,974	118	37,547
KEGG	1,090,830	34,260	21	939,258
Drugbank	517,023	19,693	119	276,142
ChEBI	4,772,706	50,477	28	772,138
SP2B-10M	10,000,457	1,730,250	77	4,690,662

Table 5.1: The FedBench datasets available at http://fedbench.fluidops.net/resource/Datasets

original datasets[1] were first cleaned through the LOD Laundromat service [28], since some of them contained invalid RDF. An overview is given in Table 5.1.

Then, each dataset was published through a TPF API on a dedicated Amazon EC2 machine (2 virtual CPUs, 7.5 GB RAM) located in the US, each running their own HTTP cache.

The client can execute the same (regular) SPARQL queries as in the single-server scenario, i.e., queries without the `SERVICE` keyword. The query-mix contains the Linked Data (LD), Life Science (LS), and Cross Domain (CD) queries from FedBench, appended with the complex queries (C) by Montoya et al. [29]. The complete query-mix was ran 20 times in sequence on the public Web, accessed from a desktop computer in Belgium in order to represent realistic long-distance latency. Per executed query, the client cache was cold and the timeout was set to 5 minutes.

We compare our measurements with numbers reported by Castillo et al. [30], who tested the following SPARQL endpoint federation systems: ANAPSID [7], ANAPSID EG (ANAPSID using Exclusive Groups), FedX [11] (with a warmed-up cache), and SPLENDID [31]. Castillo et al. obtained their measurements with one client and 10 SPARQL endpoints on different machines in a fast local network. We opted to perform our tests on a public network in order to validate the federated TPF solution within the context of the Web. Measuring recall and execution time on the Web gives an indication of the practical feasibility of real-world TPF-based federation, and a comparison with measurements of the state-of-the-art on a closed network positions TPF relative to an ideal baseline without connection delays.

	TPF	ANAPSID	ANAPSID EG	FedX (warm)	SPLENDID
LD1	1.00	1.00	1.00	1.00	1.00
LD2	0.99	1.00	1.00	1.00	1.00
LD3	1.00	1.00	1.00	1.00	0.98
LD4	1.00	1.00	1.00	1.00	1.00
LD5	1.00	0.00	0.00	1.00	1.00
LD6	1.00	0.00	0.00	1.00	1.00
LD7	1.00	0.00	0.09	1.00	1.00
LD8	1.00	1.00	1.00	1.00	1.00
LD9	1.00	1.00	1.00	1.00	1.00
LD10	1.00	1.00	1.00	1.00	1.00
LD11	1.00	1.00	0.00	1.00	1.00
LS1	1.00	1.00	1.00	1.00	1.00
LS2	0.99	0.88	0.88	0.88	0.88
LS3	0.24	1.00	1.00	1.00	1.00
LS4	1.00	1.00	1.00	1.00	1.00
LS5	0.99	1.00	0.00	1.00	1.00
LS6	1.00	0.00	0.00	1.00	1.00
LS7	1.00	1.00	1.00	0.09	1.00
CD1	0.99	0.97	0.97	0.95	0.97
CD2	1.00	1.00	1.00	1.00	1.00
CD3	1.00	0.60	1.00	0.80	0.60
CD4	1.00	0.00	0.00	1.00	1.00
CD5	1.00	0.00	0.00	1.00	1.00
CD6	1.00	0.00	0.00	1.00	1.00
CD7	1.00	0.00	0.00	0.50	0.50
C1	0.02	1.00	0.00	0.00	0.02
C2	0.93	1.00	0.00	0.00	1.00
C3	1.00	0.00	0.00	0.00	1.00
C4	0.00	1.00	0.00	0.00	0.00
C5	1.00	1.00	0.00	0.00	1.00
C6	0.77	1.00	1.00	0.00	1.00
C7	0.00	1.00	0.00	0.00	0.00
C8	0.01	1.00	0.00	0.00	0.00
C9	0.01	0.00	0.00	0.00	0.00
C10	0.00	0.00	0.00	0.00	0.00
# queries					
= *1.00*	22	21	14	20	24
≥ *0.90*	27	23	15	21	26
≥ *0.10*	30	24	16	24	30
> *0.00*	32	24	17	25	32

Table 5.2: Recall of FedBench query execution on the TPF client/server setup tested on the public Web compared to SPARQL endpoint federation systems (timeout of 300s). All occurrences of incomplete recall are highlighted. The TPF-related measurements were performed in the context of this dissertation; the numbers for the other four systems are adapted from [30].

5.4.2 Recall

Table 5.2 shows the average result recall for each query. For the queries LD, LS, and CD, the TPF setup reaches a recall of 99% or 100% for all queries except LS3, which reaches only 24% before it timeouts. Most C queries have low recall (6 queries have close to 0%), except for C3 and C5 that reach 100%. In total, 13 queries have less than 100% recall, the causes of which can be assigned into three categories. First, 6 queries (LS3, LS5, C2, C4, C7, C9) time out before all results have been computed. Especially for the complex queries, this is due to the amount of data needed by the client to complete the algorithm. Second, three queries (CD1, LD2, LS2) achieve 100% in most runs, but less than 100% in some. This is likely due to the client sending many requests out at once, which can sometimes not all be answered in time. Third, 4 queries (C1, C6, C8, C10) stop before the timeout without full recall. They cause the client to send out so many simultaneous requests that these do not all complete in time, causing the client to abort the execution. The last two causes indicate the client's vulnerability to single request time-outs, leading to the omission of (partial) results. In general, the low recall on the C queries is explained by large numbers of joins (C1) and/or the presence of complex `optional` statements (C6, C7, C8, C9, C10). This requires a large number of triples from all servers, making request time-outs more likely.

Comparing these (real-Web-based) recall measurements to Castillo et al.'s results for state-of-the-art SPARQL endpoint federation systems in a fast local network [30], we observe the following. Overall, TPF and SPLENDID have the highest number of queries with > 0% and ≥ 10% recall (32 and 30 queries, respectively), TPF has the highest number of queries with ≥ 90% recall (27), and only SPLENDID has more queries with complete results (24). For the LD queries, the TPF setup is the only one not obtaining 100% recall for LD2. On the other hand, all other systems except FedX have less than 100% recall for some of the other LD queries that the TPF setup evaluates with full completion. For the LS queries, none of the systems obtain 100% recall for LS2. The LS3 query, for which TPF only has 24% recall, is evaluated completely by all others. For the CD queries, TPF has the highest recall in all cases and only has less than 100% average recall for CD1. Especially ANAPSID and ANAPSID EG score badly on CD queries. In contrast, the C queries are problematic for most systems *except* ANAPSID, which achieves 100% recall for 7 out of 10 queries. This contrast for FedX (and ANAPSID EG) with results of the other queries is related to their usage of *exclusive groups* [11], which is not always the best strategy for the complex queries [15] as estimation errors are more likely to occur. Remarkably, TPF and SPLENDID are the only ones to achieve 100% recall for C3, for which ANAPSID and FedX do not find results. Finally, none of the systems seem to obtain significant recall for C7–C10, with the exception of ANAPSID for C7 and C8.

1. https://code.google.com/p/fbench/wiki/Datasets

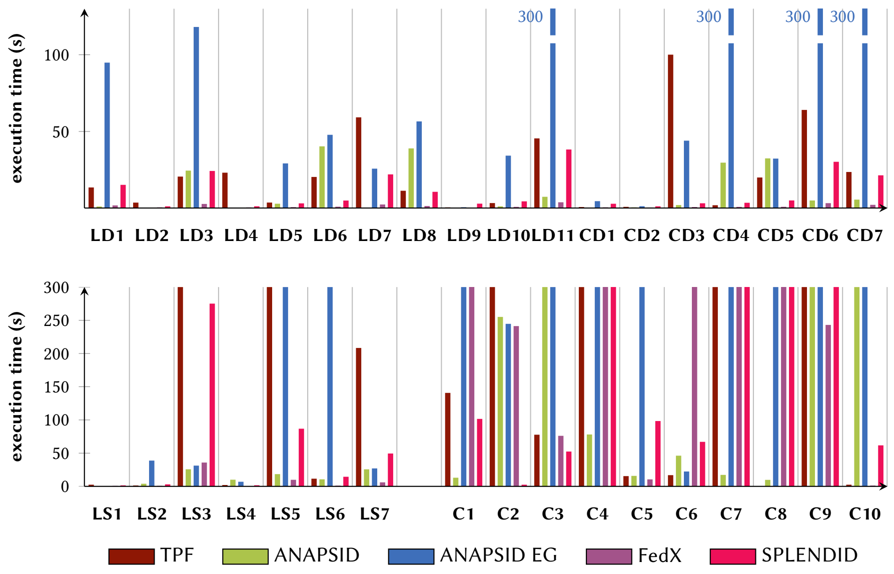


Figure 5.2: Evaluation times of FedBench query execution on the TPF client/server setup compared to SPARQL endpoint federation systems (timeout of 300s). These measurements should be considered together with the recall for each query (Table 5.2). The TPF-related measurements were performed in the context of the public Web; the numbers for the four SPARQL endpoint federation systems are adopted from [30].

5.4.3 Execution time

Next, we study the total query execution times. Our intention is to relate current query execution over TPF collections to the state-of-the-art in SPARQL endpoint federation systems. Hence, we compare to recently published results for the same queries [30].

Figure 5.2 presents the execution times for all FedBench queries, measured in seconds. The general trend is that the TPF client performs in between ANAPSID (lower bound) and ANAPSID EG (upper bound). TPF is occasionally faster than SPLENDID (LD1, LD3, and notably C5), but sometimes several times as slow (LD7, CD6, LS5, LS7). We should, however, recall that we compare against a TPF setup on a public network, so the TPF measurements include network delay. FedX with a warmed-up cache outperforms all systems for most queries, with a few notable exceptions such as C6. The timings in Figure 5.2 should, however, be considered together with the recall in Table 5.2, since not all results might have been obtained in a shorter execution time. FedX, for instance, does not find results for the C queries. A likely explanation for this difference is again FedX's usage of exclusive groups.

Considering the added network delay, a number of queries show promise: CD1, CD2, CD4, LD2, LD5, LD6, LD8, LD9, LD10, LS1, LS2, LS4, and LS6. These queries are answered in less than 4 seconds. This execution time is comparable to that of ANAPSID and SPLENDID, and even approximates FedX. A likely explanation is the presence of highly selective triple patterns in the query, for which the simplicity of TPF requests seems to compensate the overhead of query planning in other systems. For other queries (CD5, CD7, LD1, LD3), the TPF client performs worse than ANAPSID and SPLENDID, but remains within comparable bounds. A probable cause is the presence of patterns like `?x rdf:type ?y`, which can be answered by all data sources. Thus, other systems clearly benefit from prior source selection, which the current TPF client does not use.

For the remaining regular FedBench queries, the TPF client distinctly reveals its limitations. These queries (LS3, LS5) time out, or execute significantly slower (CD3, CD6, LD7, LS7) than SPARQL endpoint federation systems. They contain common predicates like `owl:sameAs` or `foaf:name` that trigger requests to all interfaces. Additionally, a high number of subject joins causes inefficiencies, since they potentially produce many membership requests to all sources, checking whether a triple is present or not. A possible enhancement is to include metadata that prevents this, such as the Approximate Membership metadata discussed in Section 4.4, at the expense of a more costly server-side interface. Another cause is the presence of a `FILTER` statement (LS7), which is currently executed client-side. This indicates room for interface extensions for such clauses [32].

The limitations of TPF become more apparent with the C queries, 4 of which time out and another 4 end prematurely (as discussed above). The high number of produced HTTP requests (3,692 on average) caused by the many triple patterns, contributes significantly to this delay. SPLENDID, FedX, and ANAPSID show similar results, but fail on different queries. For the queries where TPF reaches complete recall (C3, C5), the total execution time is comparable and even outperforms ANAPSID.

These findings, measured on the public Web, motivate a more in-depth study of complex queries for TPF, to discover possible client or server enhancements. In general, though, we notice that the performance gap observed with the single-server experiments in Section 4.3 becomes smaller in the case of federation. This indicates that the native query decomposition of TPF, combined with light requests and metadata, is more effective in federated environments, and should be examined further.

5.5 Conclusion

In a Web of Linked Data, querying federations of Linked Data sources is a necessity. Federated query processing approaches for SPARQL endpoints employ a client-server architecture where a single client queries multiple endpoints. However, these approaches are not deployable in practice due to the reliability issues endpoints face. This is currently compensated with *physical integration*, i.e., data aggregation by a central party.

Nonetheless, we argue the problem is architectural and continue to propose a shift to a *virtual integration* to consolidate data silos, i.e., composing a consumer view over distributed datasets that remain in control of the data publishers. The tendency to conflict on a policy level (e.g., challenging metadata custody and control), synchronization problems, and high infrastructural costs make physical integration particularly troublesome. Virtual integration is an attractive alternative considering the meaningful steps taken in this work to offer low-cost, *practical* solutions for many publishers. Even though at this point not all drawbacks of virtual integration can be addressed, technological advances are being made that ameliorate problems related to data source selection, uniform access for clients, and further improve maintenance costs. Triple Pattern Fragments offers low server cost and implements the client-server architecture natively; it hence offers an infrastructure more suitable to apply these advancements on than the current queryable interfaces (Research Question 3).

When tested on a public Web, the TPF client shows a competitive recall compared to the state-of-the-art SPARQL federation systems and certain queries even perform comparably—despite performance not being the main concern. Even *without* a prior source selection step; the client's run time source elimination of sources with zero-result ancestor patterns seems adequate for a number of cases. These findings validate Hypothesis 3.

Better query optimization or limited interface extensions could further improve the situation. Furthermore, an existing source selection strategy could be incorporated beforehand in order to reduce the number of considered sources but doing so might involve additional metadata and/or computations, and thereby influence the measured parameters.

This chapter was partly based on the publications:

Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. "Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web." In: *Journal of Web Semantics* 37–38 (Mar. 2016), pp. 184–206. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.03.003. URL: http://linkeddatafragments.org/publications/jws2016.pdf

Miel Vander Sande, Ruben Verborgh, Anastasia Dimou, Pieter Colpaert, and Erik Mannens. "Hypermedia-based discovery for source selection using low-cost Linked Data interfaces." In: *International Journal on Semantic Web and Information Systems* 12.3 (2016), pp. 79–110. ISSN: 1552-6283. DOI: 10.4018/ijswis.2016070103

References

[1] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. "DBpedia—A crystallization point for the Web of Data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165. URL: http://www.websemanticsjournal.org/index.php/ps/article/view/164.

[2] M Scott Marshall, Richard Boyce, Helena F Deus, Jun Zhao, Egon L Willighagen, Matthias Samwald, Elgar Pichler, Janos Hajagos, Eric Prudhommeaux, and Susie Stephens. "Emerging practices for mapping and linking life sciences data using RDF—A case series." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 14 (2012), pp. 2–13.

[3] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. "Adoption of the Linked Data Best Practices in Different Topical Domains." English. In: *International Semantic Web Conference*. 2014, pp. 245–260. ISBN: 978-3-319-11963-2. DOI: 10.1007/978-3-319-11964-9_16.

[4] Tom Heath and Christian Bizer. "Linked data: Evolving the web into a global data space." In: *Synthesis lectures on the Semantic Web: theory and technology* 1.1 (2011), pp. 1–136.

[5] Martin Klein, Robert Sanderson, Herbert Van de Sompel, and Michael L. Nelson. "Real-Time Notification for Resource Synchronization." In: *CoRR* abs/1402.3305 (2014). URL: http://arxiv.org/abs/1402.3305.

[6] Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. "A fine-grained evaluation of SPARQL endpoint federation systems." In: *Semantic Web Journal* 7.5 (2016), pp. 493–518.

[7] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. "ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints." In: *The Semantic Web – ISWC 2011*. Ed. by Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist. Vol. 7031. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 18–34. ISBN: 978-3-642-25072-9.

[8] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. "Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation." In: *The Semantic Web: Trends and Challenges*. Springer, 2014, pp. 176–191.

[9] Katja Hose and Ralf Schenkel. "Towards Benefit-based RDF Source Selection for SPARQL Queries." In: *The 4th International Workshop on Semantic Web Information Management*. SWIM '12. Scottsdale, Arizona: ACM, 2012, 2:1–2:8. ISBN: 978-1-4503-1446-6.

[10] Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. "A comparison of federation over SPARQL endpoints frameworks." In: *International Conference on Knowledge Engineering and the Semantic Web*. Springer. 2013, pp. 132–146.

[11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. "FedX: Optimization Techniques for Federated Query Processing on Linked Data." In: *The Semantic Web – ISWC 2011*. Ed. by Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist. Vol. 7031. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 601–616. ISBN: 978-3-642-25072-9. DOI: 10.1007/978-3-642-25073-6_38.

[12] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. *Vocabulary of Interlinked Datasets (VoID)*. Interest Group Note. World Wide Web Consortium, Mar. 3, 2011. URL: http://www.w3.org/TR/void/.

[13] Christoph Böhm, Gjergji Kasneci, and Felix Naumann. "Latent topics in graph-structured data." In: *The 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 2663–2666.

[14] Besnik Fetahu, Stefan Dietze, Bernardo Pereira Nunes, Marco Antonio Casanova, Davide Taibi, and Wolfgang Nejdl. "A scalable approach for efficiently generating structured dataset topic profiles." In: *The Semantic Web: Trends and Challenges*. Springer, 2014, pp. 519–534.

[15] Gabriela Montoya, Maria-Esther Vidal, and Maribel Acosta. "A Heuristic-Based Approach for Planning Federated SPARQL Queries." In: *The 3rd International Workshop on Consuming Linked Data*. Ed. by Juan F. Sequeda, Andreas Harth, and Olaf Hartig. Nov. 2012.

[16] Christoph Böhm, Johannes Lorey, and Felix Naumann. "Creating VoID descriptions for Web-scale data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 9.3 (2011), pp. 339–345.

[17] Benedikt Forchhammer, Anja Jentzsch, and Felix Naumann. "LODOP-Multi-Query Optimization for Linked Data Profiling Queries." In: *International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES), Heraklion, Greece.* 2014.

[18] Elien Paret, William Van Woensel, Sven Casteleyn, Beat Signer, and Olga De Troyer. "Efficient Querying of Distributed RDF Sources in Mobile Settings based on a Source Index Model." In: *The 2nd International Conference on Ambient Systems, Networks and Technologies.* Vol. 5. 2011, pp. 554–561.

[19] Xuejin Li, Zhendong Niu, and Chunxia Zhang. "Towards Efficient Distributed SPARQL Queries on Linked Data." English. In: *Algorithms and Architectures for Parallel Processing.* Ed. by Xian-he Sun, Wenyu Qu, Ivan Stojmenovic, Wanlei Zhou, Zhiyang Li, Hua Guo, Geyong Min, Tingting Yang, Yulei Wu, and Lei Liu. Vol. 8631. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 259–272. ISBN: 978-3-319-11193-3.

[20] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. "Comparing Data Summaries for Processing Live Queries over Linked Data." In: *World Wide Web* 14.5–6 (2011), pp. 495–544.

[21] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. "Data Summaries for On-demand Queries over Linked Data." In: *The 19th International Conference on World Wide Web.* WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 411–420. ISBN: 978-1-60558-799-8.

[22] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching.* Vol. 14. 2. ACM, 1984.

[23] Bastian Quilitz and Ulf Leser. "Querying Distributed RDF Data Sources with SPARQL." English. In: *The Semantic Web: Research and Applications.* Ed. by Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis. Vol. 5021. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 524–538. ISBN: 978-3-540-68233-2.

[24] Gio Wiederhold. "Mediators in the architecture of future information systems." In: *Computer* 25.3 (1992), pp. 38–49.

[25] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems.* Springer Science & Business Media, 2011. Chap. 9.2, p. 299.

[26] Amit P. Sheth and James A. Larson. "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases." In: *ACM Computing Surveys (CSUR)* 22.3 (Sept. 1990), pp. 183–236. ISSN: 0360-0300.

[27] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. "Fedbench: A benchmark suite for federated semantic data query processing." In: *The International Semantic Web Conference.* Springer, 2011, pp. 585–600.

[28] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. "LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data." In: *The 13th International Semantic Web Conference*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, Oct. 2014, pp. 213–228.

[29] Gabriela Montoya, Maria-Esther Vidal, Oscar Corcho, Edna Ruckhaus, and Carlos Buil-Aranda. "Benchmarking federated SPARQL query engines: are existing testbeds enough?" In: *The 11th International Semantic Web Conference*. Ed. by Philippe Cudré-Mauroux et al. Springer, 2012, pp. 313–324.

[30] Simón Castillo, Guillermo Palma, Maria-Esther Vidal, Gabriela Montoya, and Maribel Acosta. *Fed-DSATUR Decompositions*. Retrieved at 2015-09-01. 2015. URL: http://scast.github.io/fed-dsatur-decompositions/.

[31] Olaf Görlitz and Steffen Staab. "SPLENDID: SPARQL Endpoint Federation Exploiting VoID Descriptions." In: *The 2nd International Workshop on Consuming Linked Data*. Ed. by Olaf Hartig, Andreas Harth, and Juan Sequeda. Bonn, Germany, Oct. 2011. URL: http://uni-koblenz.de/~goerlitz/publications/GoerlitzAndStaab%5C%5C_COLD2011.pdf.

[32] Joachim Van Herwegen, Laurens De Vocht, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Substring Filtering for Low-Cost Linked Data Interfaces." In: *The 14th International Semantic Web Conference*. Ed. by Marcelo Arenas et al. Oct. 2015.

Don’t make me run! I’m full of chocolate!

— *Uter*

Discovering interfaces 6

A client needs to discover all the Linked Data interfaces in a federation before it can query these interfaces. Even though data source discovery has a strong impact on selecting sources that contribute to the query results, federated query execution research only discusses this marginally. This seems like a missed opportunity with all the links and semantics RDF *datasets have to offer. Therefore, this chapter addresses Research Question 4 by introducing a discovery approach for Linked Data interfaces based on hypermedia links and controls, and applies it to query execution over a federation of Triple Pattern Fragments interfaces. In addition, we identify quantitative metrics to evaluate this discovery approach (Hypothesis 4). We assess to what extent our discovery method facilitates the source selection process.*

In literature, the story of federated query execution is typically told from *source selection* onwards [1, 2]: given a fixed set of available data sources, a client determines which of these are necessary to obtain results. After that, the actual *query execution* against the selected sources happens. However, before any of this can take place, candidate data sources need to be located first. This *discovery* process preceding source selection has hardly received rigorous scientific study so far.

In general, *Linked Data source discovery* is the process of finding available Linked Data sources that are relevant to a certain task, for specific definitions of “relevance” and “task”. Although the description of dataset or endpoint characteristics has been covered, the act of *finding*, *accessing*, and *processing* such documents is still in its infancy. With an emerging Web of Linked Data, studying autonomous Linked Data discovery becomes a need, with a special focus on the impact on client-side tasks such as querying. For federated query execution in particular, discovery can assist in a more complete selection of accessed data sources.

Existing discovery works have greatly progressed in closed, custom P2P networks (e.g., Distributed Hash Tables) using custom discovery protocols [3], or centralized repositories that crawl metadata from different sources [4]. However, with a scale-free HTTP network at our disposal, little of its benefits have been exploited for Linked Data querying. Linked Data interfaces are scattered across a Web connected by hypermedia (see Section 2.1). Hypermedia allows such interfaces to function similarly to a webpage, providing users with guidance on what type of content they can retrieve, or what actions they can perform, as well as the appropriate links to do so. Since the beginning of the Web, this has been the crucial aspect to its scalability. Thus, we proposed an approach that reuses hypermedia and the Linked Data principles [5] to

- discover one another, aided by links in their dataset; and
- inform the client at run-time about their discoveries through hypermedia.

First, we investigate an approach where Linked Data interfaces locate each other *(a)* by actively following links to others, and *(b)* as a reaction to being discovered. This process is supported by *dataset summaries*, exchangeable metadata that contains high-level information on a dataset's structure, and URIs. Opposed to the methods in Chapter 4, this metadata is consumed *indirectly*—clients do not process the metadata themselves—by querying clients through constructed links.

Second, we also study the effects of source discovery on federated query processing. Both processes are not necessarily independent; an effective discovery approach benefits the source selection stage it precedes. Source selection can be impacted on several aspects, such as completeness, accuracy, and execution time. However, we are also interested in the server load, as this impacts the HTTP response times *during* query execution.

Although *any* API that provides client access to Linked Data is considered, we implemented and evaluated the approach against the lightweight Triple Pattern Fragments interface. To appropriately evaluate such a discovery approach, we introduce a methodology to quantify its parameters in Section 6.2. This includes metrics to express the functional and non-functional characteristics of one discovery approach relative to others. First, however, we layout the relevant related work next.

6.1 Discovery of Linked Datasets and Web Services

The discovery of Linked Data datasets is not a topic unknown to literature. However, the amount of existing work, and the practical deployment thereof, is still very limited. Most work (especially in standardization) focusses on description techniques through *dataset vocabularies*, which allow discovering datasets without having to access their actual interfaces. This chapter does not include any new direction in this field, but employs a custom vocabulary that is specifically selected for the proposed discovery method. This vocabulary introduced later on, after we describe the existing alternatives and their limitations first.

Vocabularies themselves do not define *how* interfaces are discovered, but often include an elementary method of access, which we discuss next. After that, we discuss the discovery of Linked Data datasets *during* query execution, as some approaches combine both processes. Finally, we also briefly discuss the discovery of Web services, which has so far been studied more deeply than the discovery of Linked Data.

6.1.1 Dataset vocabularies

Several vocabularies exist to describe and locate datasets, and some of them are W3C recommendations. The VoID vocabulary [6] enables descriptions of an RDF dataset's characteristics. It contains concepts to describe general metadata (e.g., licenses, name, or author), access metadata (e.g., SPARQL endpoint, data dump URI, or lookup URIs), and structural metadata (e.g., patterns, the dataset partitioning according to used classes or properties, and vocabularies statistics). In addition, one can describe the relations with other datasets using a *linkset*. A VoID description is applicable to many fields, including query federation, data catalogs, and faceted search applications. Unfortunately, VoID is not sufficiently fine-grained to serve applications that require efficiency, like federated query algorithms or the discovery approach presented in this chapter. For instance, one major limitation is the inability to describe common URI patterns for subjects and objects independently.

A more interface-oriented vocabulary is the SPARQL Service Description [7], which is part of the W3C SPARQL 1.1 recommendation. It covers a high level description of supported features by the endpoint and other characteristics useful to clients, i.e., default graph, entailment regime and so on. As its name suggests, this vocabulary only applies to SPARQL interfaces, not to the other interfaces discussed in Section 2.5 or any LDF instance. Therefore, it cannot be used in a uniform discovery solution.

The DCAT vocabulary is used for the description of *data catalogs*. Data catalogs are centralized indexes or repositories that contain dataset metadata. DCAT contains high-level metadata for such catalogs (e.g., title, licenses, version, etc.) and datasets (e.g., keywords, language, etc.). Datasets can be discovered by querying the indexes for certain characteristics in this metadata, and extracting their location from the results. Despite its conception as a generic vocabulary, specific elements like `dcat:byteSize` make DCAT mostly targeted toward data dumps, and not queryable Web interfaces; hence it suffers from the same limitation as SPARQL Service Description. Furthermore, since DCAT only contains detailed catalog information, discovery becomes a two-

Popular catalogs are DataHub[a] (which uses DCAT), LOV (for vocabularies)[b], re3data[c] (Registry of Research Data Repositories), and datacatalogs.org, a catalog for catalogs.

a. http://datahub.io/
b. http://lov.okfn.org/
c. http://www.re3data.org/

step process: once the catalog has been found, another process has to discover the actual dataset. This is helpful for a metadata aggregator, but not for an efficient query or discovery process.

6.1.2 Vocabulary-related discovery strategies

Vocabularies describe *what* can be discovered, but not *how* this happens. VoID suggests two methods for discovering descriptions: *(a)* via backlinks from Linked Data documents using the `void:inDataset` or *(b)* by deriving URIs to VoID descriptions canonically with a /.well-known/ URI (which is constructed with a fixed template defined in the specification). Paulheim and Hertling [8] surveyed the state in discovering VoID descriptions. They concluded that, in 85% of cases, the SPARQL endpoint could not be obtained from a Linked Data document backlink. The /.well-known/ mechanism has the best coverage (74% of URIs), but only 14% resulted in a usable description. Using dataset catalogs such as DataHub results in a higher precision for retrieving an endpoint—0.48 compared to 0.19 for /.well-known/, but they are not adopted on a large scale, suffer from datasets struggling to implement the VoID standard, and depend on the implementation of redirection.

The idea behind SPARQL Service Description is to return the description of an endpoint on a lookup of its base URI (typically /sparql/). A study by Buil-Aranda et al. [9] observed that only 35.4% of 427 endpoints registered on DataHub responded with `200 OK`, and only 11.9% returned RDF. This low uptake prevents effective usage for discovery purposes.

In this chapter, we tackle discovery by relying on the widely adopted fundamental components of the Web, i.e., hypermedia and HTTP, rather than introducing a predefined complex contract that needs to be implemented by both sides.

6.1.3 Discovery during querying

Related techniques are found in *focused Web crawling* [10], where context graphs guide link traversal based on topics.

Other strategies can be found in *link-traversal-based querying*, where discovery is integrated in the query execution. Hartig [11] identifies the *live exploration* method, which starts from a certain seed, i.e., a URI from the given query or given parameter, and recursively looks up URIs based on links. This *follow-your-nose* discovery is able to query priorly unknown sources and can potentially explore the entire Web of Linked Data (assuming sufficient connectedness). However, its recursive nature causes discovery and query execution to be slow. To improve performance, live exploration is augmented with indexes that can provide seed URIs that can shortcut exploration, or guide the exploration process. For instance, Ladwig and Tran [12] added a ranked list of URIs that is retrieved from an index.

The method in this chapter is based on similar principles, but deliberately separates part of the discovery from query execution. Thereby, it improves efficiency by *(a)* supporting a range of more complex interfaces such as TPF, and *(b)* exploiting reuse between clients by prefetching relevant interfaces in advance.

6.1.4 Discovery of services

The discovery of a Web service functionality is different from Linked Data discovery: a Web service typically offers a handful of operations, whereas Linked Data interfaces can contain millions or even billions of triples. The process of (Semantic) Web service discovery was defined by Klusch [13] as locating existing services that are relevant for a given request based on the description of their functional and non-functional semantics. Below are three categories of service discovery approaches, as identified by Klusch.

Directory-based service discovery Directory-based discovery can be centralized and decentralized. Centralized discovery depends on registries containing descriptions. Sousuo [4] collects descriptions using metasearch in existing search engines combined with focused topic crawling, to offer free text and keyword search. Decentralized approaches use a structured peer-to-peer (P2P) network and query routing protocol. For instance, AGORA-P2P [14] exploits a Chord ring to distribute the storage and location of services. An example from federated query processing is Atlas [3], which uses Distributed Hash Table (DHT) to improve source selection.

Directory-less service discovery Directory-less service discovery approaches in P2P networks include algorithms such as *flooding*, *k-random walks*, and *probabilistic adaptive search*. RS2D [15] combines probabilistic adaptive search with OWLs, such that each peer maintains a local view of the network's semantic overlay. Directory-less approaches mostly focus on popular services and provide incomplete recall.

Hybrid approaches Finally, hybrid approaches work both in structured and unstructured P2P networks. METEOR-S by Verma et al. [16] uses a set of service providing and consuming peers, which are grouped on a certain topic and domain, with a central matchmaking super-peer, which maintains a global registry with the service registry concept taxonomies of all peers.

These approaches use very interesting techniques for Linked Data source discovery, but they are all developed under the assumption of a controllable P2P network. Their appliance is unclear when translated to a scale-free and HTTP-bound Web. While they might scale well in terms of performance, they are likely to fail in terms of interoperability. Thus, in this dissertation, we choose to remain as close to HTTP as possible. However, we do recommend future research on these techniques in the context of HTTP to improve discovery.

6.2 Quantifying the discovery process

In order to study our discovery process, we need to determine what parameters can and should be measured. This then allows to compare them in a qualitative and quantitative way. In particular, it allows us to identify those parameters whose optimization can or should be the focus of algorithmic design.

6.2.1 Defining discovery

On the Web of Documents, hyperlinks connect documents into a single global information space. The target consumers are *humans*, who interact with these documents through a browser. They answer queries by using search engines, following links, and interpreting text. This process always has a clear starting point, i.e., the first URL typed in the browser (e.g., google.com), but no predefined end point. Humans *discover* relevant locations during the query solving process, while seamlessly transitioning between different Web interfaces.

Linked Data similarly connects different data sources into a single global data space, and its intended consumers are *human and machine agents* based on Web standards and the common RDF data model. On the Web, the location of all possible data sources cannot be known beforehand, let alone stored in one place. Since *"anyone can say anything about anything"* [17], a specialized discovery method is necessary to find sources to answer queries on a Web scale. We call such a method a *Link Data Interface Discovery* process.

Definition 13 (Linked Data Interface Discovery). Given a set of Linked Data interfaces I, Linked Data Interface Discovery $LDID(I, T)$ is the automated process of locating the set of Linked Data interfaces $I' \subseteq I$, whose datasets contain triples that potentially contribute to the results of a task T.

On the Web, the complete set of interfaces I is not known before, during, nor after the discovery process. However, results gathered in closed large-scale simulated environments, where I is known, can be generalized to a subweb, i.e., Web of interfaces used by a certain application, or provide indications for the entire Web.

Practically speaking, the output of an $LDID$ process is a set of URIs that identify Linked Data interfaces in a Web-like environment, as well as a certain list of characteristics for each of those interfaces.

In the context of federated query processing, the task T is the execution of a set of queries Q. Ideally, a query application also requires only one starting point, even though several servers might be required to find the desired answer to a query. A federated query algorithm then applies source selection to the output of a discovery process in order to select the relevant interfaces for a particular query. In contrast to discovery, source selection is typically repeated for each query.

6.2.2 Measuring discovery

To conclude this section, we define several parameters to measure to what extent the discovery method has succeeded. We discuss a *system of measurement* for the functional requirements, the *discovery outcome*, and measures for the non-functional requirements, the *discovery process*.

Measuring outcome

A discovery process ideally strives for *completeness*: finding all interfaces that can contribute to the results of a certain task. Thus, there is a direct dependency of the task result completeness on the completeness of a discovery process. For instance, in federated query processing, the more interfaces we are able to discover, the higher our chances of finding all possible answers, i.e., reaching completeness. Interfaces can be called *relevant* to the executed queries, if they provide access to data that contribute to the result.

Definition 14 (Relevant Linked Data Interface)**.** A Linked Data Interface $i \in I$, publishing a dataset D, is relevant to the query Q if a non-empty subset of D is used to compose a result of Q.

The completeness and accuracy of the discovery outcome can be measured with the default *recall* and *precision* metrics in closed-world experiments. However, in an open world such as the Web, the total number of relevant interfaces is unknown, since *(a)* data is distributed over a huge scale-free network, where counting datasets is infeasible; *(b)* this network is subject to constant change. Therefore, closed-world experiments performed on different scales should provide an indication on Web-scale performance.

Measuring the process

The performance marks of an *LDID* process are evaluated according to the objective of a task and the process characteristics, as defined earlier. This creates a different optimal *trade-off* mix of parameters for each process. As a result, they are incomparable with a single global performance measure. Instead, we discuss the space of non-functional requirements of *LDID* in three quantifiable aspects.

Bandwidth usage: Discovery processes need to communicate with Web resources over HTTP. Depending on their greediness, they can fundamentally differ in the amount of used bandwidth. This is measured in *number of requests* sent to complete the discovery.

Discovery execution time: Depending on the task at hand, a discovery process can be fast or slow. For example, in live exploration querying, the query execution time is directly dependent on the amount of time it took to gather the set of relevant sources. In contrast, data analysis is often performed offline, after the data is discovered and gathered. Discovery execution time is measured as the *amount of time* until the completion of the discovery process.

Server response overhead: When a task is executed, a client relies on servers to retrieve data or to provide a service. Depending on the discovery process, the impact on the default server behavior will be different. For instance, if the client handles the complete discovery, the server overhead will be zero. However, if the server is part of the process, e.g., Web crawling, a certain percentage can be added to its response time. This is measured as the *difference in time* between server responses with and without discovery.

6.3 Hypermedia-based discovery approach

In this section, we introduce a hypermedia-based discovery approach to benefit federated query clients during execution. Servers discover relevant interfaces using two methods:

Active discovery: servers dereference links from the interface data to discover relevant interfaces.

Reactive discovery: servers react to being accessed by clients, i.e., other discovering servers or querying clients.

This combination allows the discovery of outgoing links (active), as well as incoming links (reactive). Both methods rely on three foundations: *data summaries*, publishing data through a *Triple Pattern Fragments* server, and applying the *Linked Data principles*. They make two assumptions: a dataset has *one main domain authority* (e.g., http://dbpedia.org/ for DBpedia), and the owner of that domain handles the *dereferencing* of its URIs (e.g., the DBpedia pages). Both are supported by the lasting need for authoritative sources of information, which yields the sense of URI ownership—data providers are unlikely to adopt external URIs due to control concerns. Semantical disputes on the interpretation of context, where one party does not agree with the properties that are attached to a resource by another party, also hinder URI adoption. This issue already causes the non-trivial `owl:sameAs` problem [18]: a single concept ends up being identified by multiple URIs, which all need to be explicitly linked as identifiers of the same concept.

In the following, we first describe dataset summarization, then discuss both active and reactive discovery methods in more detail.

6.3.1 Creating data summaries

The discovery process is performed by an application hosting a Linked Data interface, which publishes the dataset D (as defined in Section 3.2). Our approach uses a dataset's URIs $\mathcal{U}_D$ as a seed. Thereby, the application will first create a dataset summary from D in order to access these required URIs efficiently. The motivation for this is twofold: as Linked Datasets are potentially huge, efficiently scanning an RDF dataset for URIs is fundamental, and metadata about datasets is exchanged between servers.

Such summaries are a lightweight form of *dataset profiling* and represent crucial dataset information in a compact manner. As discussed in related work, they are common in federated query processors. Given the constraints above, the desired characteristics are therefore:

- a **high summarization rate** for decreasing scan time and exchange time over HTTP;
- a **low complexity** for little server load and fast computation;
- **quick and accurate matching of triple patterns** for the hypermedia construction discussed later on.

Based on these requirements, we reuse the lightweight summaries from the query federation system HiBISCuS [19] (mentioned in Section 5.2), which are based on grouping URI *authorities* of subjects and objects per distinct predicate. In contrast to their work, we will not use the summaries for query optimization on the client, but instead let the server *(a)* iterate over foreign domains in the datasets and *(b)* generate specific hypermedia to other interfaces to inform the client.

Definition 15 (Authority function)**.** An authority function extracts the authority part, standardized by Dürst and Suignard [20], of an RDF term, and is defined as:

$$\text{authority}(u) = \begin{cases} \text{protocol}(u) + \text{hostname}(u) & \textbf{if } \text{isUri}(u) \\ \texttt{nil} & \text{otherwise} \end{cases}$$

The HiBISCuS summaries are generally very compact—the FedBench datasets (Table 5.1) are reduced to approximately 1% of their size, which is efficient for exchange and storage. Furthermore, they contain information about all terms of a triple, which makes them candidates for efficient and accurate triple pattern matching.

Saleem and Ngomo [19] include the protocol in the URI authority although this is not the case according to the standard [20].

A data summary ds is defined as a set of capabilities $\{cap_1, \ldots, cap_n\}$. For a dataset D, a capability cap_x is a triple (p, SA, OA) for a specific predicate p. Herein, SA is the set of distinct subject URI authorities of p in D, and OA the set of distinct object URI authorities of p in D. When p = `rdf:type`, an exception is made where OA is the set of distinct object URIs (instead of their authorities). We can attach a data summary to the metadata of a fragment. An example for the Jamendo dataset (from FedBench) is given in Listing 6.1.

6.3.2 Active discovery with dereferencing

This subsection details an algorithm in which a server discovers other interfaces through URIs from its own dataset.

```
@base <http://dbtune.org/fragments>.
@prefix void:  <http://rdfs.org/ns/void#>.
@prefix hydra:  <http://www.w3.org/ns/hydra/core#>.
@prefix ds:  <http://semweb.mmlab.be/ns/summaries#>.
@prefix mo:  <http://purl.org/ontology/mo/>.
@prefix foaf:  <http://xmlns.com/foaf/0.1/>.

<> hydra:member <#dataset>.
<#dataset> a void:DataSet, hydra:Collection;
 ds:capability [
  ds:predicate foaf:based_near;
         ds:sbjAuthority <http://dbtune.org/>;
         ds:objAuthority <http://sws.geonames.org/>
  ];
 ds:capability [
  ds:predicate rdf:type;
  ds:sbjAuthority <http://dbtune.org/>;
        ds:objAuthority foaf:Document, mo:MusicArtist
 ];
 ds:capability [
  ds:predicate foaf:name;
        ds:sbjAuthority <http://dbtune.org/>;
 ].
```

Listing 6.1: The fragment http://dbtune.org/fragments#dataset has its metadata extended with a dataset summary.

The Jamendo dataset (Table 5.1) contains 776,611 distinct subjects and objects, but is covered by 655 samples because there are only 655 unique authorities.

Phase 1: Identify external servers The discovering server actively tries to identify other servers based on data summaries, the Linked Data principles, and hypermedia. This process is *data-driven*: it exploits the links to external datasets, which are possibly hosted by other servers. Therefore, we start by identifying a set of *foreign* URIs $\mathcal{R} = \{r_1, \dots, r_n\} \subseteq \mathcal{U}_D$ in the dataset (with $\mathcal{U}_D$ as the set of distinct URIs in dataset D; see Section 3.2). Foreign URIs differ in hostname with the interface, thus their lookup is handled by other servers. We rely on the data summary generation process to populate $\mathcal{U}_D$ with one sample URI per authority for efficiency reasons.

Phase 2: Dereference entities on external servers The discovering server acts as a *client* and discovers other servers by dereferencing each $r_x \in \mathcal{R}$. In these responses, it looks for triples (r_x, `rdfs:isDefinedBy`, u_{f_x}) (Figure 6.1a). The URI u_{f_x} possibly identifies a TPF f_x of the dataset that contains r_x. If no such triple is found, r_x does not support this discovery approach, and the discovery for this URI ends here.

Continuing our previous example, the Jamendo server on domain dbtune.org could dereference the foreign URI http://sws.geonames.org/660013/. This lookup is subsequently answered by the server on sws.geonames.org, which hosts the Geonames dataset as Linked Data documents. The response contains the triple

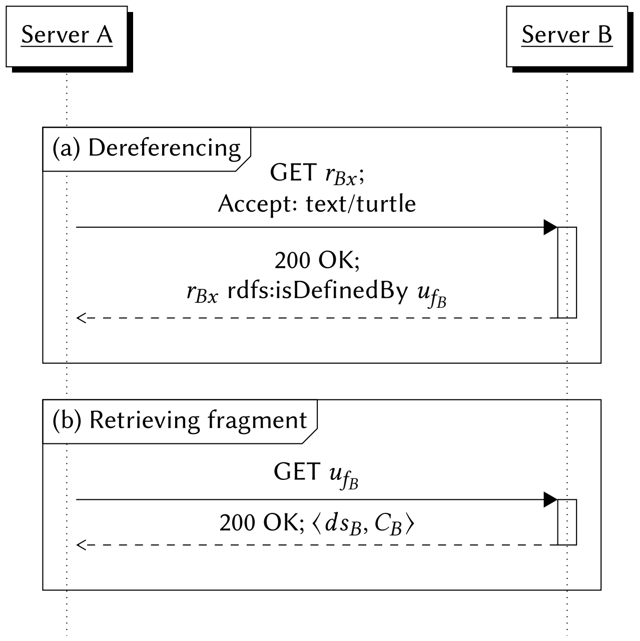


Figure 6.1: Active discovery discovers Linked Data interfaces by dereferencing foreign URIs in the dataset and retrieving the linked Triple Pattern Fragment.

(http://sws.geonames.org/660013/, `rdfs:isDefinedBy`, http://sws.geonames.org/fragments/), where the object identifies a TPF of the Geonames dataset.

Phase 3: Identify controls and summaries The server requests f_x and looks for hypermedia controls C_x and a data summary ds_x in the response (Figure 6.1b). The hypermedia controls reveal whether the resource is a TPF, and if so, allow requesting other TPFs of the dataset. The data summary informs the discoverer about the data that can be found in the interface, which is used later on to construct hypermedia to this server. For the fragment http://sws.geonames.org/fragments, an extract of the HTTP request cycle is shown in Listing 6.2. The data summary starts at line 12 and the controls at line 18.

Phase 4: Index storage The tuple $\langle ds_x, C_x \rangle$ is stored in an index I_{ds} with u_{f_x} as key. To prevent the same fragments to be requested multiple times, the presence of f_x is looked up first in the set of keys. Note that in some cases, e.g., a cache expiry, it is desirable to re-request a fragment to update its summary and controls.

Active discovery ends when all sample URIs have been dereferenced, so I_{ds} contains of list of discovered summaries.

```
GET http://sws.geonames.org/fragments HTTP/1.1
Accept: text/turtle
Referer: http://dbtune.org/fragments
--------------------------------------------
HTTP/1.1 200 OK

<http://sws.geonames.org/fragments>
 hydra:member
   <http://sws.geonames.org/fragments#dataset>.
<http://sws.geonames.org/fragments#dataset>
  a void:Dataset, hydra:Collection;
  ds:capability [
  ds:predicate gn:parentFeature;
    ds:sbjAuthority <http://sws.geonames.org/>;
    ds:objAuthority <http://sws.geonames.org/>
  ];
...
hydra:search _:triplePattern.
_:triplePattern hydra:template
  "/fragments{?subject,predicate,object}";
hydra:mapping _:subject, _:predicate, _:object.
_:subject hydra:variable "subject";
  hydra:property rdf:subject.
_:predicate hydra:variable "predicate";
  hydra:property rdf:predicate.
_:object hydra:variable "object";
  hydra:property rdf:object.
```

Listing 6.2: A fragment of the discovered interface returns hypermedia controls and a data summary.

6.3.3 Reactive discovery with the Referer header

Since active discovery is limited by the foreign URIs present in the dataset, its reach is insufficient for two reasons. First, queries that require knowledge about backlinks cannot be answered, leading to low result recall. Since hyperlinks are unidirectional, servers can only discover interfaces that host their outgoing links. For example, the query given in Listing 6.3 needs both triples from Jamendo and Geonames. However, as Geonames has no links to Jamendo, Geonames's index would not contain Jamendo, and is therefore not able to retrieve the triples necessary to yield results. Second, when a dataset changes, its summary in the index of other servers will be outdated. Therefore, a server needs to notify those who have discovered it about this change.

Referer is actually a misspelling of the word "referrer", but remained unnoticed until after the standard was released.

Reactive discovery tackles both issues by using the HTTP `Referer` header from incoming requests, inspired by the work of Mühleisen and Jentzsch [21], to automatically create new links in the LOD cloud. `Referer` is standardized in RFC2616[1], which states:

1. http://tools.ietf.org/html/rfc2616#page-140

> The Referer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained

In this approach, two actors are equipped with this header:

Querying clients: a query execution application supplies the last accessed fragment, which enables the discovery of servers with no explicit link in their dataset;

Discovering servers: a server doing active discovery supplies a random fragment URI (in practice the index fragment) on which it can be discovered in turn.

The process happens as follows:

Phase 1: Header inspection When a server receives a request by client, it checks whether a Referer header is present. If it is, it should be a URI u_{f_x} (Figure 6.2a). Possibly, u_{f_x} identifies a Triple Pattern Fragment f_x.

Phase 2: Dereference the referrer The server verifies that u_{f_x} is a foreign URI with an authority it has not (recently) visited before. If so, the server dereferences URI u_{f_x} to obtain the resource f_x.

Phase 3: Identify controls and summaries The server inspects f_x in order to find the summary ds and controls C_x, analogous as in Phase 3 of the active discovery process. For example, given the request in Listing 6.2, the Geonames server can react by requesting the referrer URI http://dbtune.org/fragments given in line 3. As a result, it retrieves the summary and controls for the Jamendo server.

Phase 4: Index storage If both are correctly extracted from the response, the tuple $\langle ds_x, C_x \rangle$ is stored in an index I_{ds} as well, with u_{f_x} as key.

Since HTTP caches can intercept the Referer header, they could prevent reactive discovery—and caching is a crucial scalability factor for Triple Pattern Fragments. However, the cache could maintain a list of referrers (for instance, in server logs), which are then passed to the server at a later stage.

6.4 Federated query processing through hypermedia

In this section, we describe how clients consume the result of a discovery process for the execution of queries. Clients rely on HATEOAS (hypermedia as engine of application state; see Section 2.1), where they react autonomously to links included in HTTP responses. First, we discuss how the hypermedia are constructed on the server. Second, we describe how the client consumes the links to retrieve more relevant fragments on the fly. As a running example,

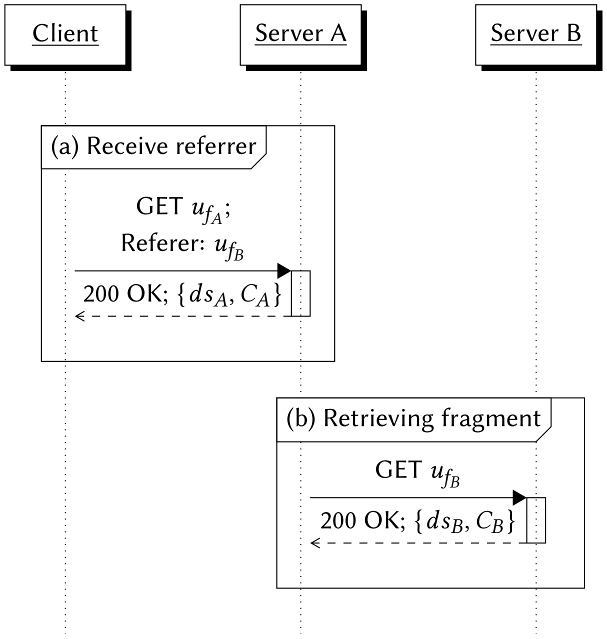


Figure 6.2: Reactive discovery discovers interfaces by retrieving fragments from the Referer header.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX gn:<http://www.geonames.org/ontology#>

SELECT ?name ?location ?news WHERE {
  ?artist foaf:name ?name .
  ?artist foaf:based_near ?location .
  ?location gn:parentFeature ?germany .
  ?germany gn:name "Federal Republic of Germany"
}
```

Listing 6.3: An example query combining the Geonames and Jamendo datasets

we discuss the evaluation of the query in Listing 6.3 over the Geonames and Jamendo data sources mentioned in the previous section.

6.4.1 Building the server-side hypermedia interface

When a fragment is requested, the server informs the client about the interfaces it has discovered in the response. Instead of sending all known interfaces, it pre-selects the ones most relevant to the requested fragment in order to aid with source selection. This reduces the chance of source overestimation on the client.

When handling a request, the server has the following data at its disposal:

1. the requested triple pattern $tp = (s, p, o)$, extracted from the request;
2. an indexed set of data summaries $I_{ds} = \{u_{f_1} \Rightarrow \langle ds_1, C_1 \rangle, \ldots, u_{f_n} \Rightarrow \langle ds_n, C_n \rangle\}$.

Based on their summaries, the server looks for servers that can possibly answer the requested triple pattern as well. Therefore, a server iterates over all summaries in I_{ds}. For each summary, it checks for a match using the algorithm in Listing 6.4.

```
Data: triple pattern $tp = (s, p, o)$,
      data summary $ds = \{cap_1, \ldots, cap_n\} \in I_{ds}$
Result: $m(tp, ds) \in \{\text{true}, \text{false}\}$
1  $capabilities \leftarrow \begin{cases} \{ds[p]\} & \textbf{if } isUri(p) \\ ds & \text{otherwise} \end{cases}$;
2  for $cap = (p, SA, OA) \in capabilities$ do
3      return false if $cap = \texttt{nil}$;
4      $s_{auth} \leftarrow authority(s)$;
5      $o_{auth} \leftarrow authority(o)$;
6      return true if $s_{auth} = \texttt{nil} \wedge o_{auth} = \texttt{nil}$;
7      $match_s \leftarrow (s_{auth} = \texttt{nil} \vee s_{auth} \in SA)$;
8      if $p = \texttt{rdf:type}$ then
9          $match_o \leftarrow (o_{auth} = \texttt{nil} \vee o \in OA)$;
10     else
11         $match_o \leftarrow (o_{auth} = \texttt{nil} \vee o_{auth} \in OA)$;
12     end
13     return true if $match_s \wedge match_o$;
14 end
15 return false;
```

Listing 6.4: Determine if a given triple pattern matches a data summary, and possibly its dataset.

First, we select the summaries to iterate over (line 1). If the predicate is a variable, we select all summaries in the index. If the predicate is a URI, we look up its entry. Next, we iterate over the selected set of capabilities. If no entry is found, the summary does not match (line 3). For each summary, if the subject or object is a URI, we extract its authority (line 4,5). If it is not a URI, it's either a variable, blank node or literal, which always match (line 6). Finally, a server with a data summary only has a chance of matching the requested triple pattern if

1. the predicate is `rdf:type`, which is an exception in the data summary, and the object in the pattern is an exact match of a URI in the object authority (line 9); or
2. the subject is a variable or its subject authority is equal to the authority of the pattern's subject, and the same holds for the object (line 11).

For instance, when the Jamendo server receives an HTTP request for the *tp* on line 7 of Listing 6.3, it returns zero triples, as the dataset does not contain

```
<http://dbtune.org/fragments?object=http%3A%2F%2Fwww.geonames.org%2Fontology%23
     parentFeature> rdfs:seeAlso
<http://sws.geonames.org/fragments?object=http%3A%2F%2Fwww.geonames.org%2Fontology
     %23parentFeature>.
```

Listing 6.5: Although the Jamendo server returned no triples for the pattern tp = $(?x,$ `gn:parentFeature`, ?y), it contains a hypermedia control to the corresponding fragment in the Geonames dataset.

`gn:parentFeature`. Before responding, it scans the index for a match, which contains the summary for http://sws.geonames.org/fragments. As shown in line 13 of Listing 6.2, the Geonames summary does contain `gn:parentFeature` and is a match, since the subject and object of tp are variables. Similarly, when the tp on line 5 of Listing 6.3 is requested from the server, it returns triples, because `foaf:name` is present in the dataset. However, the Geonames summary does not match, since `foaf:name` is not present.

For each fragment u_{f_x} that matches the requested triple pattern ($m(ds_x, tp)$ = true), we create a direct link URI $u_{f'_x}$. This link points to the fragment for the same triple pattern on a remote server. The resulting set of links $H = \{u_{f'_m}, \dots, u_{f'_n}\}$, with $n \geq m \geq 0$ and $|H| \geq 0$, are the hypermedia controls of the response, i.e., we add zero or more `rdfs:seeAlso` links[2] to inform the client about the other servers. For instance, if the Geonames summary matches the triple pattern in Listing 6.3, line 7, the hypermedia triple in Listing 6.5 will be added to the fragment's response. Then, the client can follow this `rdfs:seeAlso` link to collect (more) matching triples.

6.4.2 Client-side query execution

SPARQL queries can be federated over multiple TPF servers with the TPF client from Section 4.2 in combination with the mediator layer from Section 5.3. In the following, we alter the mediator layer to consume the links to relevant fragments offered by the server.

The client starts with one fragment URI, instead of a list, but accesses more TPF interfaces *during* execution. The mediator still aggregates multiple fragments from different servers using function $\vartheta(tp_j) = \sum_{i=1}^{N} \text{cnt}_{ij}$ (with tp_j as the requested triple pattern) and exposes them as a single fragment. However, it will first follow the `rdfs:seeAlso` links present in the first fragments's response by calling a recursive function.

To illustrate this process, we execute the query in Listing 6.3 against the Jamendo server from earlier. For each triple pattern, the corresponding fragment is retrieved from the server which results in:

2. http://www.w3.org/wiki/UsingSeeAlso

```
?artist foaf:name ?name.                          # 3,505 matches
?artist foaf:based_near ?location.                # 3,244 matches
?location gn:parentFeature ?germany.              # 0 matches
?germany gn:name "Federal Republic of Germany".  # 0 matches
```

Typically, the algorithm would stop the execution here; the last two patterns do not match any triples and therefore lead to an empty result set. However, the response for both patterns does include a link to a corresponding fragments on the Geonames server. Now, the algorithm retrieves each link, appends the result to the original fragment result set and sums the count metadata with ϑ. Now, both fragments do return matches, which updates the previous results:

```
?artist foaf:name ?name.                          # 3,505 matches
?artist foaf:based_near ?location.                # 3,244 matches
?location gn:parentFeature ?germany.              # 0 + 7,479,713
                                                  # = 7,479,713 matches
?germany gn:name "Federal Republic of Germany".  # 0 + 1 = 1 match
```

After aggregation, the last triple pattern has the lowest number of matches, namely 1. Thus, the algorithm continues with the single matching triple for this pattern and binds the value http://sws.geonames.org/2921044/ to the variable `?germany`. Finally, the process is repeated for the remaining three triple patterns.

6.5 Experiments

The experimental results presented in this section aim to provide insight in the performance of the hypermedia-based discovery approach and its impact on federated query execution. We describe the implementation, test setup, and performed tests as follows.

6.5.1 Experimental setup

We implemented our approach as a plugin for the existing Triple Pattern Fragments NodeJS server. This plugin is called the *Explorer* and runs in a separate single-threaded process, *asynchronous* to the server application, indexing data summaries as they arrive. The server communicates with the Explorer using three methods for *(a)* starting the active discovery; *(b)* passing a `Referer` header if detected in a request; and *(c)* passing a triple pattern from the request to retrieve a set of `rdfs:seeAlso` links, i.e., the hypermedia controls.

To test the query execution, we used the FedBench [22] benchmark and its corresponding collection of datasets. We selected the Cross Domain (CD), Life Science (LS), and Linked Data (LD) queries from the list of pre-defined queries, as they make use of multiple servers. Queries LS7 and CD6 are not supported by our implementation, and are therefore excluded from the results.

In total, we allocated 9 virtual machines as servers in Amazon EC2 Virtual Private Cloud, one for each dataset. Each virtual machine has the equivalent of a dual-core Intel Xeon E5-2670 v2 CPU, 7.5 GB of RAM and 32 GB of SSD storage. The datasets KEGG and ChEBI were merged to avoid dereferencing conflicts, as they share the same authoritative URI hostname. Each server was provided with an active NGINX[3] HTTP cache and a TPF server including the Explorer module, both handling the dereferencing and the TPF interface. Its `/etc/hosts` file was edited to simulate the different domain names. The specific FedBench dataset was added to the TPF server as an HDT data source.

An extra virtual machine was created to run the FedBench client and gather server logs. A plugin for a Triple Pattern Fragments client supporting both hypermedia-based and naive federation was added to the FedBench client.

6.5.2 Feasibility of the discovery approach

In the following, we assess to what extent a discovery approach is feasible by using hypermedia. First, we inspect the data summary construction, which has the biggest impact on the server. Next, we expect the number of required HTTP requests to have a major impact on the discovery execution time. Fortunately, this number is directly dependent on the amount of links (indicated by a triple's object) to other datasets, which is only a small fraction of all URIs present. Therefore, the discovery execution time should scale well with the size of the dataset, and bandwidth usage should be limited.

Finally, we assess the completeness of our discovery approach. The active discovery is data-driven, which should cover all outgoing links, while reactive discovery should cover all incoming links. Hence, all directly relevant interfaces should be discovered.

Summary complexity

We measured the summary generation time for each dataset, which is shown in Table 6.1. Each summary was constructed from its corresponding HDT file. The results confirm that the summaries are lightweight in terms of complexity and size.

The largest dataset, i.e., Geonames with 108,000 triples, was summarized in only 10.85 minutes, while the smallest dataset, i.e., Semantic Web Dog Food, was summarized in less than a second. These low computation times can partially be explained by the highly efficient HDT format, which offers a highly-compressed browsable read-only index. However, using such data source is defendable. Opposed to supporting SPARQL, the TPF interface encourages servers to adopt more limited, but highly specialized lightweight indexes. Furthermore, no multi-threading was used during generation, which can still be fully exploited and compensate slower data sources, e.g., writable indexes like X-RDF-3X [23] or SPARQL endpoints. Another factor is likely the sole use of URI

3. http://nginx.org/

authorities, which can be derived with a single inexpensive string operation (e.g., a highly efficient regular expression operation).

Next, we look at the generation time and its relationship to the anatomy of the dataset. In particular, we investigate whether the generation time correlates with the number of distinct subject, predicates, and objects from Table 5.1. Their associations are plotted in Figure 6.3.

Presumably, the distinct predicates have little influence as they are highly outnumbered by the distinct subjects and objects. For instance, Geonames and DBpedia have the highest generation times, while DBpedia has 30 times more distinct predicates than Geonames. Therefore, a Spearman's rank-order correlation was run to determine the relationship between the number of distinct predicates and the summary generation time. There is a weak, monotonic relationship between the two variables ($\rho = 0.0425$), which can not be considered statistically significant with a two-tailed p(-value) $= 0.90708(\alpha = 0.05)$. Hence, the role of the number of predicates in the complexity of summary generation is not clear.

Now we explore the possible dominance of the number of distinct subjects and objects. In particular, we want to know whether the time to generate summaries sufficiently scales, i.e., predictable linear growth is ensured, when datasets grow larger. A performed Spearman's rank-order correlation tested the relationship between the total number of distinct subjects and objects, and the summary generation time. We observe a strong, positive monotonic relationship ($\rho = 0.988$), which is statistically significant with a two-tailed $p < 0.0001(\alpha = 0.05)$. Moreover, the solid trendline in Figure 6.3 suggests a linear relationship between the two variables. This was confirmed using a Pearson product-moment correlation. There is a strong, positive linear correlation between the total number of distinct subjects and objects, and the summary generation time with statistical significance, indicated by $r = 0.999$ with two-tailed $p < 0.0001(\alpha = 0.05)$.

Finally, we study the influence of dataset size, i.e., the total number of triples. The dataset size is expected to have a similar correlation as subjects and objects because of their dominance. In Figure 6.3, the dashed trendline already indicates a linear relationship between the total number of triples and the summary generation time. To confirm this, a Spearman's rank-order correlation was run. There is a significant strong, monotonic relationship between the total number of triples and the summary generation time given by $\rho = 1.000$ with a two-tailed $p < 0.0001(\alpha = 0.05)$. Additionally, a similarly strong, linear correlation is indicated by a Pearson product-moment correlation with $r = 0.993$, which is also statistically significant with $p < 0.00001(\alpha = 0.05)$.

The correlation findings above in combination with the overall low execution times show that the data summaries can be computed in a reasonable time, which is suggested to be most impacted by the size and more specifically, the number of subjects and objects in the dataset. Improving the handling of these

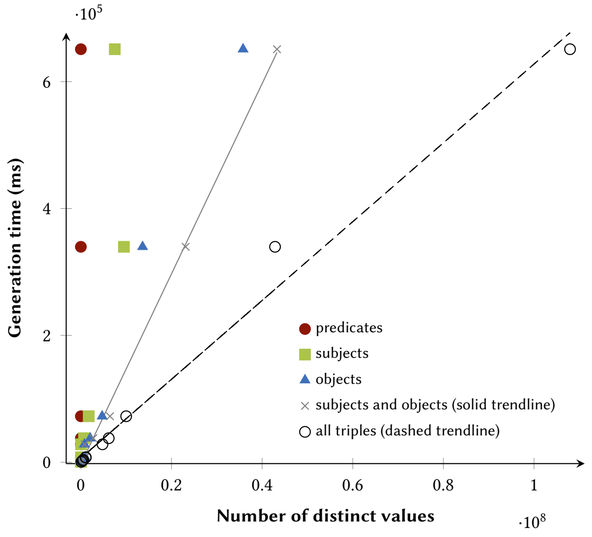


Figure 6.3: The data summary generation time increases linearly with the number of distinct subjects and objects.

values instead of the predicates, seems more rewarding for optimizing summary generation.

Summary size

A dataset summary should be sufficiently compact to ensure that *(a)* the summary index is able to scale, and *(b)* summaries can be transferred over HTTP efficiently. Consequently, we aim for the summaries to be at least 95% smaller than the original dataset.

The fourth column in Table 6.1 shows the summarization ratio per dataset. All values lie above 99% compared to the already compact binary HDT file. They have a sample mean $\mu = 99.612$ and a standard deviation $\sigma = 0.990$.

Shapiro-Wilk's test indicates that the summarization ratio scores are not normally distributed ($p < 0.01$). Therefore, a one-tailed Wilcoxon signed-rank test for single sample was performed to test whether the mean summarization ratio is larger than 95%, which yielded a W(-value) = 0. The critical value of W for $N = 10$ at $p \leq 0.05(\alpha = 0.05)$ is 8 and the Z(-value) = $-2,666$ with p-value = $0,0038(\alpha = 0.05)$. Hence, the data summaries are significantly small ($\mu > 95$; $W < 8$).

Because of the URI authorities, summarizing the dataset creates a lot of redundancy, which allows us to greatly compress the result. The created redundancy decreases according to the number of distinct authorities present in the dataset. This potentially leads to a low summarization ratio for highly variable datasets (e.g., sameAs.org). However, as previously pointed out, this is rare be-

dataset	generation time (ms)	JSON size (kB)	ratio (%)	original HDT size (kB)
DBpedia subset	339,598	85.68	99.98	584,679
NY Times	2,237	25.12	99.77	11,344
LinkedMDB	37,840	26.10	99.95	52,413
Jamendo	7,390	53.69	99.70	18,258
Geonames	651,102	26.01	99.99	1,035,674
SW Dog Food	667	83.17	96.80	2,606
KEGG	7,755	1.62	99.98	11,850
Drugbank	3,650	13.27	99.92	16,942
ChEBI	28,233	1.87	99.99	23,502
SP2B-10M	72,482	6.38	99.99	320,662

Table 6.1: The compression rate of data summaries is very high (99%) with acceptable creation time (11 min)

cause of URI ownership, where data publishers tend to base their URIs around a single domain name.

In terms of absolute size, the summaries remain below a few hundred kilobytes. The largest summary, i.e., DBpedia, responsible for roughly 40 million triples, results in only 85 kB. Although this size increases slightly when loaded into memory, we can conclude that they are sufficiently compact. The server can load many summaries in memory without introducing large overhead. Furthermore, the current implementation only applies a simple URI dictionary, thus, a more extensive compression can still massively reduce space.

Discovery execution time and bandwidth usage

To evaluate the performance of our hypermedia-based discovery approach, we deployed 8 servers, one for each dataset. We excluded the SP2B dataset since it uses local URIs and is not linked to any dataset. Each server was provisioned with a pre-computed data summary and the server application. The active discovery process was executed on all machines simultaneously, also triggering the reactive discovery.

Table 6.2 shows the results for the active discovery phase. All servers were able to discover their direct neighbors, of which three (DBpedia, LinkedMDB, Drugbank) in less than 10s, four (NY Times, Jamendo, Geonames, SW Dog Food) in less than 7min and one outlier (KEGG–ChEBI) in only 5ms. This outlier is due to the absence of external URIs in the KEGG–ChEBI dataset, which requires no requests. The difference between the two other clusters also correlates with their differences in request count. Thus, higher bandwidth usage reflects on a higher execution time. The highest execution time was measured for the SW Dogfood dataset. Although it has the lowest number of triples, the maximum of 771 requests was sent. This can be explained by its high number of links to personal author websites, resulting in a high number of unique samples. In general, the low number of requests is a result of using a data summary as seed. Despite the large number of triples in a dataset, e.g., DBpedia, the request count can be very low (e.g., 16 requests), due to low authority variance in its URIs.

Next, we analyze the influence of bandwidth usage and dataset size on the discovery execution time. A Spearman's rank-order correlation was run to determine the relationship between the number of requests and the discovery execution time. There is a strong, positive monotonic relationship that is statistically significant given by a $\rho = 0.952$ with a two-tailed p(-value)$= 0.00027(\alpha = 0.05)$. Furthermore, a Pearson's product-moment correlation test with $r = 0.943$ and p(-value)$= 0.00044(\alpha = 0.05)$ shows a strong, positive linear correlation, which is also statistically significant. This is confirmed by a coefficient of determination $R^2 = 0.889$.

The number of HTTP requests sent, however, is only a fraction of the dataset size (i.e., the total number of triples). The average request per triple is only 0.1% ($\sigma = 0.001$). Yet, a Spearman's rank-order correlation test shows a weak, negative monotonic relationship between the dataset size and the discovery execution time with $\rho = -0.38095$, which is not statistically significant with a two-tailed p(-value)$= 0.35181(\alpha = 0.05)$. We observe a similar result with Pearson's $r = -0.3402$ and coefficient of determination $R^2 = 0.1157$, which indicate a weak negative linear correlation that is statistically insignificant with a two-tailed $p = 0.409926(\alpha = 0.05)$. Hence, this suggests that the number of HTTP requests could be a more suitable predictor than the dataset size.

Finally, Table 6.3 shows the results of the reactive discovery phase. Each server has discovered the interfaces that were connected through a backlink. Reactive discovery is cheap in terms of bandwidth usage, as it only needs one requests per discovery. As a result, its execution times are mostly determined by time taken to reconstruct the summary from the response, as shown in the last two columns.

Completeness of the discovery outcome

In terms of completeness, the combination of active and reactive discovery results in a recall of 1.0 ($\mu = 1.0$), as all interfaces relevant to the discovery task are found. Here, a relevant interface provides access to a dataset to which a link is included in the discoverer's dataset.

If the number of stored summaries deviates in Table 6.2, it was already discovered by reactive discovery, or visa versa. During active discovery, both DBpedia and SW Dog Food found links to OWL or RDFS, but these were detected as not being fragments. Completeness in context of a query task is discussed in the next section.

dataset	links found	#l	#f	#s	#req	sct (ms)	et (ms)
DBpedia	OWL NY Times	2	1	1	16	41	1,073
NY Times	DBpedia Geonames	2	2	1	240	13	30,666
LinkedMDB	DBpedia Geonames	2	2	2	15	118	6,704
Jamendo	Geonames	1	1	1	655	31	218,549
Geonames	DBpedia	1	1	1	261	129	24,853
SW Dog Food	OWL DBpedia Geonames Drugbank	4	3	3	771	211	382,787
Drugbank	RDFS DBpedia KEGG–ChEBI OWL	4	2	2	22	126	7,609
KEGG–ChEBI	–	0	0	0	0	–	5

Table 6.2: Active discovery is able to find interfaces for all links. No links are found for KEGG–Chebi, since it does not contain any outgoing links. (#l=number of links, #f=number of fragments, #s=number of fragments stored in index, #req=number of sent requests, sct=summary construction time, et=execution time)

6.5.3 Impact on federation query execution

In this second experiment, we measure the impact of our discovery approach on client-side federated query execution. We ran FedBench on different setups, combining a client variant and a specific index. Two variants of clients were implemented: *(a)* a naive client that sends every requests to every server, without consuming hypermedia, and *(b)* a hypermedia-based client which implements the query execution algorithm described in this chapter. The naive client will act as a baseline to measure the impact of the hypermedia-based client. It mimics existing federation systems that solely rely on triple patterns to do both source selection and query execution. We do not use other state-of-the-art federation systems as baseline, since the comparison would be unfair. Such systems are highly tailored to the efficient, but computationally heavy SPARQL interface, which enables very low execution times. Our efforts aim at a sustainable solution for both client and server, thus using the slower, but lightweight TPF interface. We aim at investigating the role of hypermedia in source-selection, rather than optimizing execution speed.

In addition, we prepared two distinct index provisions: *(a)* populated with all summaries of the other datasets (*full index*), and *(b)* populated with the sum-

dataset	links found	#f	#s	#req	asct (ms)	aet (ms)
DBpedia	Geonames Drugbank SW Dog Food, LinkedMDB NY Times	4	4	1	56	73
NY Times	DBpedia	1	1	1	123	142
LinkedMDB	–	0	0	–	–	–
Jamendo	–	0	0	–	–	–
Geonames	LinkedMDB NY Times SW Dog Food, Jamendo	4	4	1	49	53
SW Dog Food	–	0	0	–	–	–
Drugbank	–	1	1	1	36	76
KEGG–ChEBI	Drugbank	1	1	–	30	48

Table 6.3: Reactive discovery appends the backlinked interfaces to the active discovery results. (#f=number of fragments, #s=number of fragments stored in index, #req=number of sent requests, asct=avg. summary construction time, aet=avg. execution time)

maries from its discovery outcome (*discovery index*). The full index setup provides insight on the general performance of hypermedia-based source selection. It creates the ideal scenario where each interface is able to provide a link to any other interface. This isolates the query execution from any limitations of the discovery approach, i.e., incomplete indexes. The discovery index setup provides insight in the relation between discovery outcome and the query algorithm. Here, each interface has a different, more selective index, which enables studying the impact on query completeness and execution time.

In total, this second experiment addresses query result completeness, server overhead, and query execution time. First, since we expect the recall of the discovery outcome to be high, the hypermedia controls should be sufficient to produce high query result recall as well (> 0.75). Second, we expect a limited impact of the hypermedia control generation process on the server. The summaries are very compact, directly available in the main memory, and contain all information for efficiently producing the links, given the algorithm described in Section 6.4. Third, given that hypermedia can be generated at low-cost, sending hypermedia to the client should result in a more selective source selection. These benefits are unlikely to outweigh the extra overhead.

Query result completeness

In the following, we study the completeness and accuracy of the discovery outcome from Table 6.2 and Table 6.3.

	DBpedia	NY Times	LinkedMDB	Jamendo	Geonames	SW Dog Food	Drugbank	KEGG-ChEBI
CD1	1.00	0.86	0.86	0.00	ERR	0.86	0.86	0.00
CD2	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD3	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD4	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD5	0.00	0.00	ERR	0.00	ERR	0.00	0.00	0.00
CD6	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
CD7	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LS1	1.00	0.00	0.00	0.00	ERR	0.00	1.00	1.00
LS2	1.00	0.00	0.00	0.00	ERR	0.00	0.96	0.96
LS3	–	–	–	0.00	ERR	–	–	–
LS4	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LS5	–	–	–	0.00	ERR	–	–	–
LS6	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD1	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD2	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD3	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD4	1.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD5	0.00	1.00	1.00	ERR	ERR	1.00	1.00	0.00
LD7	0.32	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD8	–	–	–	0.00	ERR	–	–	–
LD9	0.00	0.00	0.00	0.00	ERR	0.00	0.00	0.00
LD10	0.00	1.00	1.00	0.00	ERR	1.00	1.00	0.00

Table 6.4: The recall of each query's results depends on the starting dataset. Due to only using links one level deep, only interfaces with the relevant interfaces as direct neighbors contribute to recall.

A FedBench run was performed with each server as starting point. In terms of completeness, Table 6.4 shows the measured recall of each query for each run, with the number of results from the naive client run as baseline. In general, many queries returned no results, which results in zero or undefined recall (naive client returned 0 results, marked with "–"). Empty result sets are likely caused by the limitations of the current client, which only consumes the hypermedia controls of one fragment response, i.e., the direct neighbors of that particular server. Therefore, the client cannot retrieve data from interfaces that are more than one link away from the starting point. Given these findings, we clearly need a more powerful execution algorithm to reach a better coverage in recall.

Using DBpedia as starting point reaches 1.0 recall for most of the queries, because its index also contains summaries for most datasets. Moreover, queries LD5 and LD10 contain predicates from DBpedia, and, thus can be answered by using starting points with links to the DBpedia summary. However, this means the DBpedia server itself should also be a good starting point, but this is not reflected in the results. This likely is caused by an error during execution, e.g., a failed HTTP request.

The low recall scores demand a more intelligent client algorithm that can deal with multiple hops, while keeping query execution scalable. Another first enhancement would be to define heuristics to estimate the levels of neighboring links to traverse. While it is unfeasible to blindly traverse interfaces, this would potentially consume the whole Web, a possible improvement is a dynamic hop count based on, for instance, the number of joins in a query. The client could also rely on link traversal during execution to reach non-indexed interfaces that are out of immediate reach.

Server response overhead

We tested the impact of our discovery approach on a server responding to a querying client. As stated above, a client solves complex queries by retrieving a series of fragments from a server. Hypermedia is added to each response, thus the overhead is measured by its share in hypermedia construction time. To not endanger scalability, only an overhead below 10% is considered acceptable.

Being implemented as a simple JavaScript object, the summary index should eventually be replaced with a more efficient data structure.

Figure 6.4 shows the average HTTP response time per server. All servers respond within 3ms on average, confirming the lightweight character of Triple Pattern Fragments servers in combination with an HDT data source. A left-tailed t-test was run to determine whether hypermedia control construction is below the 10% accepted response overhead. Hypermedia construction scores are normally distributed, as assessed by Shapiro-Wilk's test ($p > 0.05$) and there are no outliers in the data. A mean hypermedia construction overhead of 5.24% ($\sigma = 0.67\%$) is reported to be lower than a hypermedia construction overhead of 10%. This is a statistically significant difference of 4.76% with t-value $= -20.094$ and $p < 0.0001 (\alpha = 0.05)$. For a single request, the overhead of constructing addition hypermedia can therefore be considered negligible.

Query execution time

First, we measured whether using summary indexes and hypermedia decreases the average execution time. Figure 6.5 shows the mean execution time for the

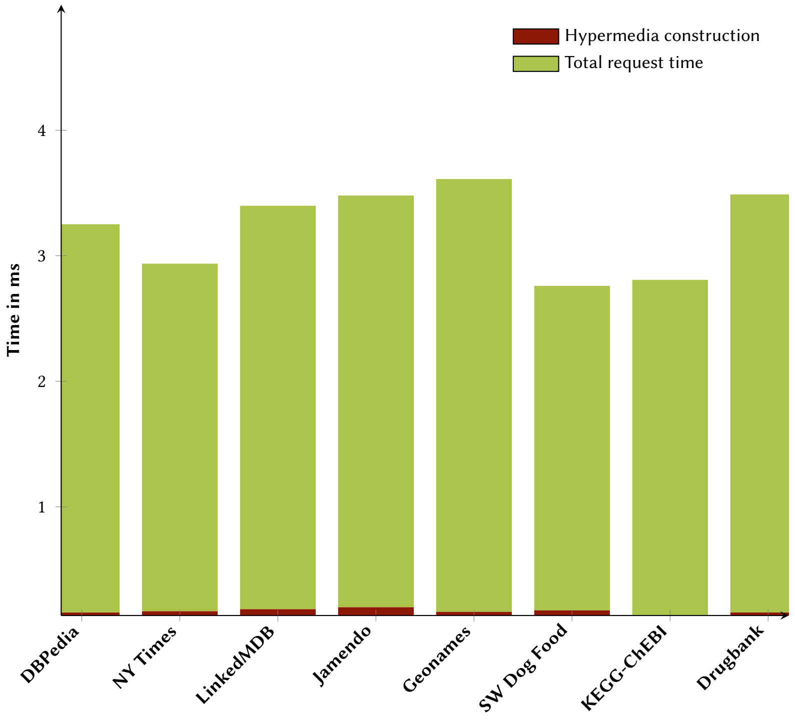


Figure 6.4: Hypermedia construction time is negligible compared to the total response time.

naive client on index-free servers and the *hypermedia-based client on full index servers*. All results achieved full completeness (recall = 1.0), and are ordered descending according to the number of results.

The score differences between the naive and hypermedia-based client are not normally distributed, as assessed by Shapiro-Wilk's test ($p < 0.01$). Hence, a one-tailed Wilcoxon signed-rank test was performed with Z-value= -1.9642 ($N \geq 20$) and $p = 0.025(\alpha = 0.05)$). This shows that using a hypermedia-based client on servers with a full index, inflicts a statistically significant decrease in mean execution time compared to using naive clients on index-free servers. The difference in mean execution time is 10.817s over all queries.

For most queries, we notice a small performance increase for the hypermedia-based client, caused by its more selective nature. However, the queries LS3, CD3, and LD4 show an decrease in mean execution time of 50% or more, which indicates impact. They contain a triple pattern with a URI bound to the object, which is very selective. This enables the indexes to be very selective for one or more generic patterns, i.e., `?x rdf:type ?y`, `?x owl:sameAs ?y`, or `?x foaf:type ?y`. This strictly reduces the amount of requests sent, in con-

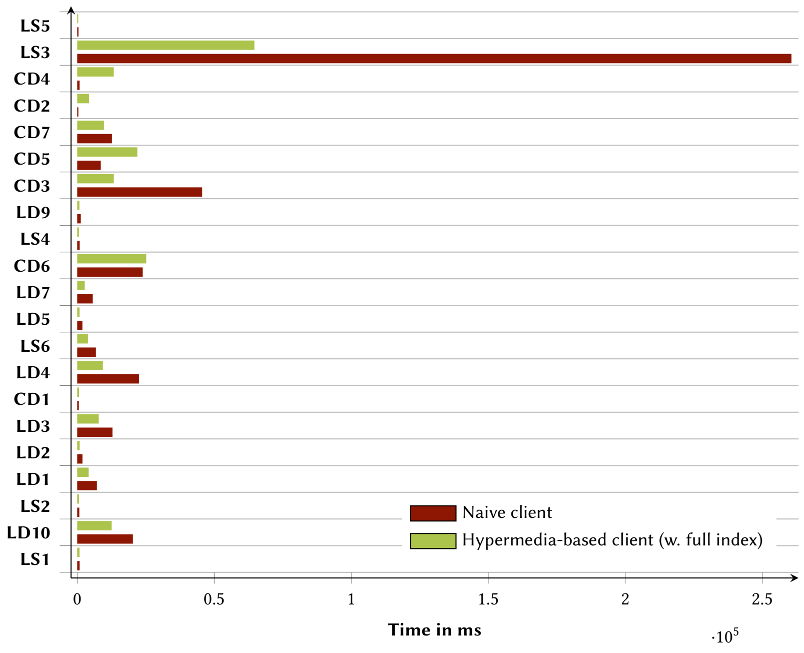


Figure 6.5: Hypermedia-based querying with full-index servers decreases the mean execution time with 50% compared to the naive federation approach

trast to the naive client, which keeps accessing all servers. The query CD6 shows a slight decrease in performance. Since its only bounded object is a literal, the index cannot be selective at first, as literals are not included in the summaries. Thus, the hypermedia overhead outweighs the gain on future iterations of the query algorithm.

Next, we compared the execution time for the *discovery index* setup with the *full index* setup. From Table 6.4, we selected the queries that were answered with 1.0 recall by at least one starting point. If there are multiple starting points for one query, we chose the one with the highest execution time. The results are shown in Figure 6.6.

Shapiro-Wilk's test indicates that the score differences between a full index and a discovery index are not normally distributed ($p < 0.01$). A two-tailed Wilcoxon signed-rank test was therefore performed with W(-value) = 4($N < 20$). The critical value of W for $N = 15$ at $p \leq 0.05(\alpha = 0.05)$ is 25 and the Z(-value) = -3.1806 with a $p = 0.00148(\alpha = 0.05)$. Therefore, using a discovery index elicits a statistically significant change in mean query execution time compared to using a full index. For most queries, there is a significant decrease, especially LS3, CD4, and CD2. These queries clearly benefit from the smaller index, which only gives a few links to check.

In general, we cannot claim hypermedia-based source selection decreases the

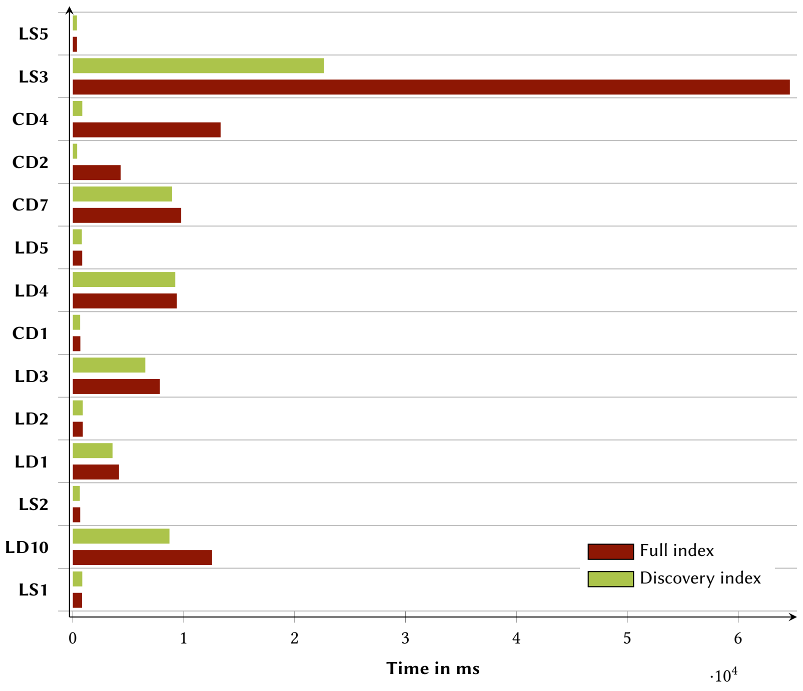


Figure 6.6: For a few queries, hypermedia-based querying with a discovery index is several magnitudes faster than with a full-index. However, most queries are only slightly faster, not yielding any claims on query time decrease.

average query time compared to the baseline with source elimination only. However, the results indicate potential for hypermedia in source selection, in particular when using more targeted indexes created by discovery. A more thorough evaluation is advised with a larger number of datasets and a more extensive query mix.

6.6 Conclusion

Query federation over low-cost interfaces is a crucial aspect for evolving the Web towards a global, machine understandable data space. For query federation systems, an important performance aspect is source selection: determining which sources are relevant to evaluate a certain SPARQL query.

However, Linked Data sources need to be discovered first, and this process potentially impacts source selection too. Therefore, we proposed a hypermedia-based approach to benefit federated query processors (Research Question 4). In addition, we defined a framework to evaluate this approach, covering the quantitative aspects of a discovery process. These quantitative aspects cover the non-functional requirements completeness and accuracy, and the functional requirements bandwidth usage, execution time, and server overhead. Existing processes using discovery can thus be formalized, enabling comparability between approaches and the development of a proper methodology. However, the categorization and metrics are high-level, and need to be studied in-depth in the future.

Just like hyperlinks fade the boundaries between different Web interfaces for humans, a discovery process should fade the distinction between *query execution* and *federated query execution*. The experimental results are promising in terms of server overhead and completeness in the pre-query phase. When executing federated queries, our approach shows a general decrease in execution time, but very poor recall. We conclude that the link traversal has been simplified, but still requires algorithms that intelligently consume hypermedia.

Overall, we have to falsify Hypothesis 4 and acknowledge further steps need to be taken to call a discovery process effective. For instance, combining query execution with additional hypermedia or metadata repositories (e.g., LODLaundromat [24], DataHub, or LOD-a-lot [25]) might be necessary to improve recall. Servers can for instance construct these repositories by dedicating themselves to the proposed discovery process, opposed to also serving data.

This chapter was partly based on the publication:

Miel Vander Sande, Ruben Verborgh, Anastasia Dimou, Pieter Colpaert, and Erik Mannens. "Hypermedia-based discovery for source selection using low-cost Linked Data interfaces." In: *International Journal on Semantic Web and Information Systems* 12.3 (2016), pp. 79–110. ISSN: 1552-6283. DOI: 10.4018/ijswis.2016070103

References

[1] Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. "A comparison of federation over SPARQL endpoints frameworks." In: *International Conference on Knowledge Engineering and the Semantic Web*. Springer. 2013, pp. 132–146.

[2] Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. "A fine-grained evaluation of SPARQL endpoint federation systems." In: *Semantic Web Journal* 7.5 (2016), pp. 493–518.

[3] Zoi Kaoudi, Kostis Kyzirakos, and Manolis Koubarakis. "SPARQL Query Optimization on Top of DHTs." In: *The 9th International Semantic Web Conference on The Semantic Web*. ISWC'10. Shanghai, China: Springer-Verlag, 2010, pp. 418–435. ISBN: 3-642-17745-X, 978-3-642-17745-3.

[4] Matthias Klusch and Xiguo Zhing. "Deployed Semantic Services for the Common User of the Web: A Reality Check." In: *The International Conference on Semantic Computing*. Vol. 8. 2008, pp. 347–353.

[5] Tim Berners-Lee. *Linked Data – Design issues*. Ed. by Tim Berners-Lee. July 27, 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html (visited on 06/18/2009).

[6] Keith Alexander and Michael Hausenblas. "Describing linked datasets – on the design and usage of VoID, the 'Vocabulary of Interlinked Datasets'." In: *The Linked Data on the Web Workshop*. Citeseer, 2009.

[7] Gregory Todd Williams. *SPARQL 1.1 Service Description*. Recommendation. World Wide Web Consortium, Mar. 21, 2013. URL: http://www.w3.org/TR/sparql11-service-description.

[8] Heiko Paulheim and Sven Hertling. "Discoverability of SPARQL Endpoints in Linked Open Data." In: *The International Semantic Web Conference (Posters & Demos)*. ISWC-PD'13. Sydney, Australia: CEUR-WS.org, 2013, pp. 245–248. URL: http://dl.acm.org/citation.cfm?id=2874399.2874461.

[9] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. "SPARQLWeb-Querying Infrastructure: Ready for Action?" In: *The 12th International Semantic Web Conference*. Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz. Nov. 2013.

[10] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. "Focused Crawling Using Context Graphs." In: *International Conference on Very Large Data Bases*. 2000, pp. 527–534.

[11] Olaf Hartig. "An Overview on Execution Strategies for Linked Data Queries." In: *Datenbank-Spektrum* 13.2 (2013), pp. 89–99.

[12] Günter Ladwig and Thanh Tran. "Linked Data Query Processing Strategies." English. In: *The International Semantic Web Conference*. Ed. by Peter. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, JeffZ. Pan, Ian Horrocks, and Birte Glimm. Vol. 6496. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 453–469. ISBN: 978-3-642-17745-3.

[13] Matthias Klusch. "Service discovery." In: *Encyclopedia of Social Networks and Mining (ESNAM)* (2014). Ed. by R. Alhajj and J. Rokne.

[14] A Basit Khan and Mihhail Matskin. "Agora framework for service discovery and resource allocation." In: *The International Conference on Internet and Web Applications and Services*. IEEE, 2010, pp. 438–444.

[15] Ulrich Basters and Matthias Klusch. "RS2D: fast adaptive search for Semantic Web services in unstructured P2P networks." In: *The International Semantic Web Conference*. Springer, 2006, pp. 87–100.

[16] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. "METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services." In: *Information Technology and Management* 6.1 (2005), pp. 17–39.

[17] Tim Berners-Lee. *Axioms of Web Architecture: Metadata*. Jan. 1997. URL: https://www.w3.org/DesignIssues/Metadata.html.

[18] Harry Halpin, Patrick J. Hayes, James P. McCusker, Deborah L. McGuinness, and Henry S. Thompson. "When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data." English. In: *The International Semantic Web Conference*. Ed. by Peter Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm. Vol. 6496. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 305–320. ISBN: 978-3-642-17745-3.

[19] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. "Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation." In: *The Semantic Web: Trends and Challenges*. Springer, 2014, pp. 176–191.

[20] Martin Dürst and Michel Suignard. *Internationalized resource identifiers (IRIs)*. Request For Comments 3987. Internet Engineering Task Force, 2005. URL: https://tools.ietf.org/rfc/rfc3987.

[21] Hannes Mühleisen and Anja Jentzsch. "Augmenting the Web of Data using Referers." In: *The Linked Data on the Web Workshop*. 2011.

[22] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. "Fedbench: A benchmark suite for federated semantic data query processing." In: *The International Semantic Web Conference*. Springer, 2011, pp. 585–600.

[23] Thomas Neumann and Gerhard Weikum. "X-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases." In: *The International Conference on Very Large Data Bases*. Vol. 3. 1-2. VLDB Endowment, Sept. 2010, pp. 256–263.

[24] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. "LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data." In: *The 13th International Semantic Web Conference*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, Oct. 2014, pp. 213–228.

[25] Javier D Fernández, Wouter Beek, Miguel A Martínez-Prieto, and Mario Arias. "LOD-a-lot." In: *International Semantic Web Conference*. Springer. 2017, pp. 75–83.

Every time I learn something new, it pushes some old stuff out of my brain! Remember when I took that home wine-making course and forgot how to drive?

— Homer J. Simpson

Accessing history 7

Content on the Web drifts and this is no different in a Web of Linked Data that is connected through persistent URIs. Changing resource states make re-executing a query in its valid context difficult. This chapter addresses this reproducibility problem by tackling Research Question 5 and proposing a sustainable Linked Data publishing strategy for RDF archives based on virtual integration. Accordingly, we discuss the necessary enhancements on (a) storage, (b) publishing, and (c) querying level (Hypothesis 5). First, for storing RDF archives in a queryable way, we discuss a pragmatic file-system-based technique that covers many common cases. Second, we extend the low-cost Triple Pattern Fragments interface discussed in the previous chapters to publish these RDF archives on the Web. A time-dimension is added to the interface to synchronize distributed sources while querying and answering queries over time. Third, we adjust the query client and federation mediator to support SPARQL querying in the time dimension. To put this into practice, this strategy is discussed in context of the Libraries, Archives, and Museums domain.

Driven by the need for persistent URIs, the Web tends to push an *eternal now* perspective. A representation returned by requesting an HTTP resource always captures the then-current state. Yet, the fact that Web content is fleeting—it disappears or changes over time—makes this quite a contradiction. Changing resource states can cause content to *drift* and create inconsistencies between resources. For instance, a Web page on average current income that serves as a reference for a page on unemployment rate in 2004, may no longer support the made claims. Essentially, as popular information source, the Web struggles with the problem of *reproducibility* [1]: query results—either by browsing or search—need to remain valid when they are regenerated later.

Evidently, as identification by persistent URIs drive Linked Data [2], this issue affects the Web of Linked Data as well. Collections of RDF statements experience factual changes (e.g., demographics or artwork portfolios), or schematic changes (e.g., use of categories, types or relations) that prevent re-executing a query in its valid context [3]. Querying a federation of drifting sources can produce invalid results due to incompatible schemas, broken links, or combining facts that are no longer true.

Thus, domains with an increased demand for Linked Data, like Cultural Heritage and Open Data, cannot consider digital preservation an afterthought. In addition, the responsible institutions are frequently under-resourced, making low-cost Linked Data publishing their particular interest. For that reason, we expand the narrative on low-cost interfaces from previous chapters to support Linked Data archiving. This yields the benefit of querying past versions of a dataset, which we can use to synchronize distributed interfaces or analyze the evolution of query results.

To keep a firm connection with the practical setting, we approach this chapter from the perspective of Libraries, Archives, and Museums (LAMs), a domain where cross-institutional data integration, with the desire for virtual integration and reproducibility, is prominent and a matter of daily concern. Before we can introduce a sustainable integration strategy though, we need a good understanding on how these institutions publish Linked Data today.

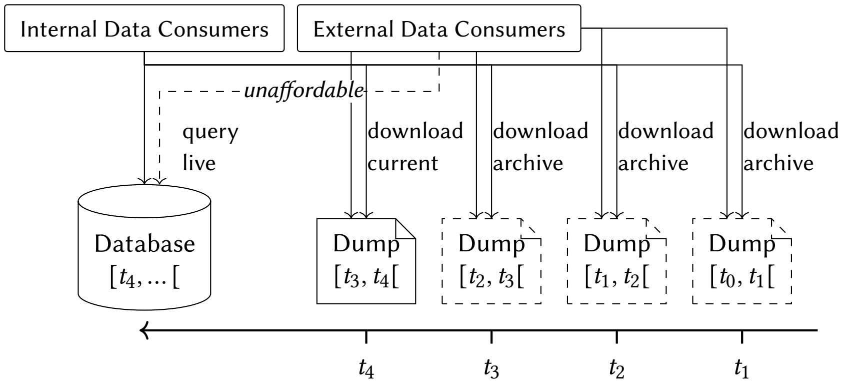
7.1 Publishing Linked Data preservation efforts

Well aware that ongoing Linked Data access can not be reliable over time *without* preservation [4], Libraries, Archives, and Museums aspire a combined agenda of long-term digital preservation and access [5, 6]. As mentioned in Section 2.4, an Extract-Transform-Load workflow periodically produces a new RDF collection. Consequently, with attention to digital preservation, a Web publishing workflow always applies to an RDF *version* of the dataset, for which three conceptually different types exist:

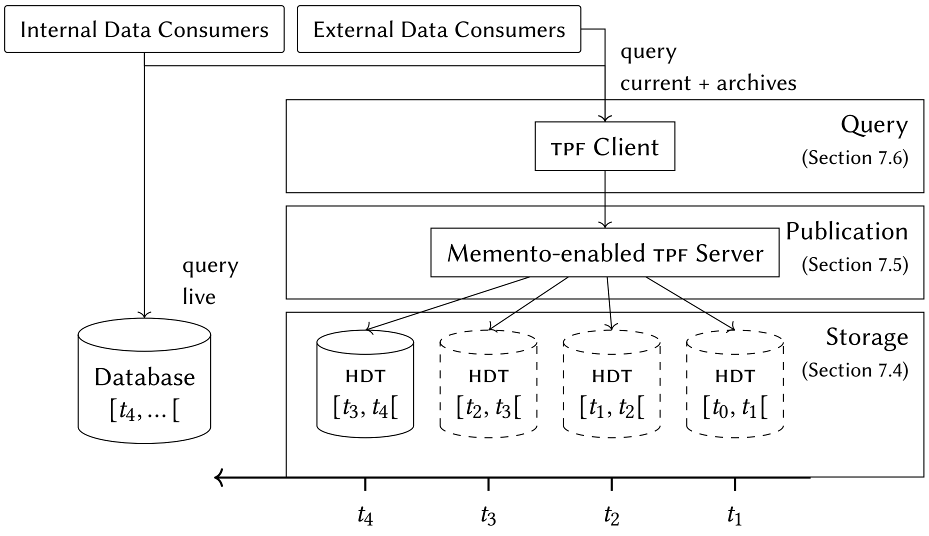
- the **live** version, representing the last possible state of the dataset, i.e., the collection currently known to the publishing institution;
- the **current** version, representing the latest *published* state of the dataset, i.e., the most recent version accessible to consumers;
- an **archive** version, representing a state of the dataset bounded by a fixed temporal range for which it is valid, i.e., preserved versions for future reference.

Typically, the *live* version is loaded into an RDF database, which offers powerful functionalities to clients, such as fine-grained querying. Yet, under-resourced organizations struggle to enhance this workflow to provide effective external access to their digital preservation efforts—the *archive* versions. They resort to the Linked Data publishing approach shown in Figure 7.1a.

Despite its usefulness, the *live* RDF database rarely serves as *current* version as well, but is restricted to internal consumption. This means the database is not exposed to external consumers and understandably so. As mentioned in Section 3.1, there is little economical justification to share complex, maintenance-intensive SPARQL systems with unpredictable load. Thus, such institutions re-



(a) Common Linked Data publishing strategy centered around data dumps



(b) TPF-centric approach with support for (HDT-based) RDF archiving

Figure 7.1: Enhancements of the proposed Linked Data publishing strategy to the common approach.

lease timely *archive* versions as data dumps and make them available for download [7]. The most recent dump serves as *current* version, and is therefore never entirely up-to-date. In the best case, multiple *archive* dumps serve as preservation strategy, making the history available for internal and/or external use. However, issues like the fear of loosened control—when shared, data can be used for anything and by anyone, might prevent publishers to keep all archives available forever. Despite its inexpensive character, this approach strongly limits functionality for the client, as the data is not directly queryable.

7.2 Sustainable publishing and querying of Linked Data archives

Given that the current impasse is motivated by a lack of funds, inspecting whether embracing the low-cost TPF interface can lead to a more sustainable strategy seems evident. Again, this pinpoints a sweet spot regarding *interface availability* by balancing maintenance costs and query functionality, but now includes both *live* and *archive* versions.

We suggest an alternative strategy to facilitate cross-institutional data integration. Hereby, we employ a *virtual integration* approach (see Section 5.1); every organization can publish their own datasets, which clients integrate on demand through means of querying. This results in the toolchain illustrated in Figure 7.1b, with focus on three main benefits for the publisher, its internal consumers, and external consumers. First, infrastructure requirements become *lower cost*, while query capabilities remain. By restricting the expressiveness of the interface, the required computational resources become more predictable. This lowers the barrier for institutions to publish their metadata collections online. Second, *uniform access* to the semantically integrated data across institutions is achieved by merely relying on the HTTP protocol and a self-describing interface. Hence, any Web application can interoperate with the datasets and clients can autonomously discover how to interact. Third, it provides *synchronization* in the sense that querying dataset versions that have matching temporality and that reside in distributed RDF Archives is possible using the same interface.

As already pointed out in Section 4.1.1, a publisher can generate HDT files relatively cheaply, while the extended lookup interface highly increases functionality for external and internal consumers.

In contrast to the approach in Figure 7.1a, we argue to keep the database, holding the *live* version, unexposed, but to recurrently publish *archive* versions of a dataset in a compact, but more expressive storage solution, e.g., the HDT format (introduced in Section 2.4.2) or an RDF version control system. This modest shift enables digital preservation in combination with access through a low-cost TPF-based wrapper stack. Complex queries can be executed in a public setting on both a *current* version, i.e., the most recently extracted indexed version, and its *archives*. Although the *current* and *live* version still do not coincide, RDF storage solutions, especially HDT, can be of lower maintenance compared to collecting data dumps.

In total, we propose to unlock this functionality by composing three layers:

1. the **Storage** layer contains techniques to archive RDF data and provide access to all versions independently;

2. the **Publication** provides a wrapper TPF interface with added support for temporal queries; and
3. the **Query** layer queries multiple temporal TPF interfaces together to answer complex queries over archives.

All layers are discussed in Sections 7.4 to 7.6, but first, we introduce a proper use case in the next section to enable a correct validation of this proposed strategy.

7.3 Reconstructing institutional history from archives

As no one size fits all, the contextual constraints in which a publishing strategy is beneficial, need clarification. Hence, we embody such constraints in a Digital Humanities use case that consists of *reconstructing memory from distributed knowledge sources.* For example, recreating the status of scientific collaboration networks or reassembling the virtual presence of institutions as they existed at some point in the past. Its presence allows us to support and validate the approach introduced in the previous section. In technical terms, in order to tackle this type of challenge, a client needs to be able to fulfill two actions:

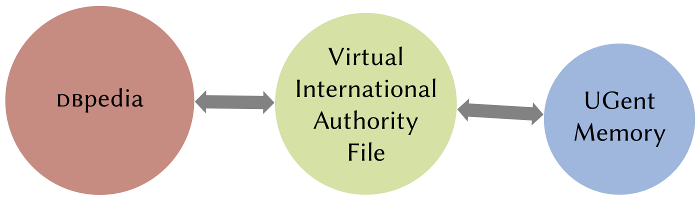
1. selecting distributed data sources in a manner that ensures that their contained knowledge temporally coincides, that is, represents the state of affairs at a same point in time;
2. querying the selected distributed data sources to retrieve the data required to reconstruct the desired memory.

As a concrete example, we selected three semantically integrated Linked Datasets (Figure 7.2) as distributed data sources. This selection of datasets is heterogeneous regarding the type of knowledge that is represented, the typical purpose the knowledge serves, and the size of the knowledge database:

DBpedia DBpedia is a *large* (> 1 billion triples) RDF dataset derived from Wikipedia. It contains an abundance of *common knowledge* facts and acts as a *general-purpose* linking hub between numerous domain-specific datasets that reuse DBpedia URIs. DBpedia also reuses external URIs including those from the VIAF authority file. DBpedia is versioned in biannual static releases.

Virtual International Authority File (VIAF) VIAF [8] is a reputable authority file that is jointly compiled by LAMs worldwide. An authority file is an independent index of authority records to relate and govern the headings used in a bibliographic catalog, thus enforcing authority control. VIAF is a *medium-sized* (10 million – 1 billion triples) dataset available in various formats, including as a Linked Dataset. It contains entries pertaining to organizations and authors, and as such qualifies as a *thematic* dataset. As a global authority file, it is a *general-purpose* dataset that is used in a wide variety of settings, including as a linking hub in the LAM community.

UGentMemorialis UGentMemorialis [9] is a *small* (< 10 million triples), *thematic*, *organization-specific* Linked Dataset that contains information about professors that worked at the University of Ghent, Belgium. It is maintained by the University Library, and serves as a prime example of a LAM institution using Linked Data to increase institutional visibility. UGentMemorialis uses VIAF URIs to uniquely identify the professors about which it holds information.



Knowledge:	common	thematic	thematic
Purpose:	*general*	*general*	*organization-specific*
Size:	large	medium	small

Figure 7.2: Our use case exists of an illustrative semantically integrated network of three organizations publishing the datasets DBpedia, VIAF, and UGent-Memorialis.

These complementary datasets are created by different authorities, and, as is typically the case with Linked Datasets, they are interlinked through the mutual reuse of URIs.

In accordance to the aforementioned client actions, we formulate four queries over DBpedia, VIAF, and/or UGentMemorialis, that each represent a common query format. Besides the sources' *current* version, we also target *archive* versions to enable memory reconstruction. Thereby, we tackle the problem of *reproducibility* caused by evolving RDF statements.

Our use case targets both *single source* and *multiple sources*, to access the contained knowledge of distributed data sources. Such queries exploit the links between multiple datasets, i.e., UGentMemorialis is linked to DBpedia via VIAF through the resources that represent authors. By combining multiple sources that support access to *archive* versions, we can ensure temporal synchronization. The resulting queries are presented below:

Query to a Single Source (DBpedia)

- Current: *What is the number of awards won by Belgian academics?*
- Archive: *How did the number of awards won by Belgian academics evolve between 2008 to 2016?*

Query to Multiple Sources (DBpedia, VIAF & UGentMemorialis)

- Current: *What is the number of DBpedia triples describing professor "Haus, Jacques-Joseph" of Ghent University?*
- Archive: *How did the number of DBpedia triples describing professor "Haus, Jacques-Joseph" of Ghent University evolve between 2008 and 2016?*

7.4 Storage solutions for RDF archives

Data access and preservation are preferably a combined effort [6], as distributed sources need to be synchronized in time when striving for the *reproducibility* of queries. Typically, the kind of queries clients want to execute determines the the publication interface, which in turn sets the requirements for RDF storage. For the purpose of cost-effectively publishing data archives, we reverse this chain and start from the storage technology, as it is the constraining factor for the remainder of the pipeline. In other words, before *current* and *archive* versions can be published on the Web and subsequently queried, they first needs to be stored in a sustainable *RDF archive*.

7.4.1 Requirements for RDF archives

Papastefanatos [10] identified the two challenges in Linked Open Data evolution management:

1. **quality control and maintenance**, semantic integration issues such as schema or URI changes;
2. **data exploitation**, ensuring valuable insights can be retrieved from the evolution itself.

RDF archive systems should engage in the latter and offer so-called LOD long term accessibility [10], where "datasets with different time and schema constraints coexist and must be uniformly accessed, retrieved and combined". Therefore, they should adhere to the following long-standing characteristics of data archives. Houghton [11] puts the **large data volume** forward as major challenge. Although the overall cost is decreasing, it is still challenging to store, maintain, and process the vast amounts of metadata today. Thus, both infrastructure and software need to efficiently cope with resource constraints. Next, Ross [12] states data in a digital library should be read-only in order to "accept it as authentic". Metadata collections need to be represented as **immutable snapshot** versions to protect them from change and corruption. Finally, to ensure synchronization, such past versions should be **immediately accessible**, i.e., browse and lookup operations on current and past snapshots need to be equally fast.

In the remainder of this chapter, we discuss a pragmatic approach based on HDT and the file system, which is sufficient for most common cases.

7.4.2 Pragmatic Linked Data archives with HDT

For many scenarios, including our publishing strategy in Section 7.2, a "Good Enough" Linked Data preservation solution [13] is sufficient. Therefore, we first discuss a pragmatic approach to RDF archiving using the file-system and the HDT format. Often, such an approach is more affordable for organizations and serves all of their needs, while avoiding any complex software system.

Figure 7.3 presents a matrix structure which organizes a collection of HDT files. The vertical axis represents the different RDF collections that are currently archived. The horizontal axis represents versions of those collections generated at prior points in time. Each version is valid for the interval between the time it was generated and when the succeeding version was generated. From the metadata in the HDT Headers, we can automatically construct a simple index that supports a *lookup*(D, t) operation. Thereby, we can retrieve a valid HDT file D_t with a given dataset name D (e.g., RDF Dataset B) and timestamp t (e.g., t_2). Every newly extracted HDT file is added to the first column, which contains the *current* version. All affected datasets shift right in the matrix, and the index is updated to reflect this change.

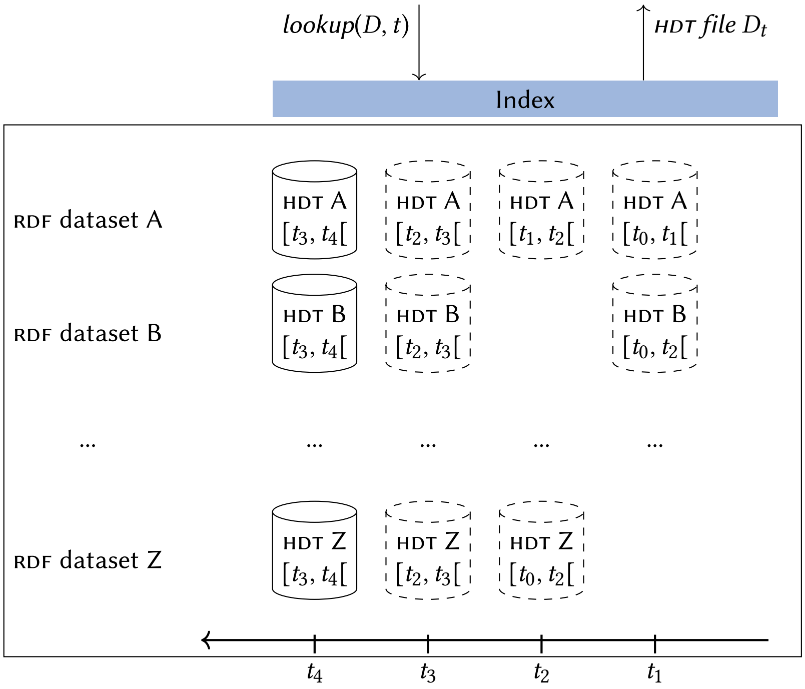


Figure 7.3: By applying HDT horizontally and vertically, we can create a simple, but efficient RDF archive. The operation *lookup*(D, t) retrieves a valid HDT file D_t with a given dataset name D and timestamp t

For validation, we re-designed the existing DBpedia archive initially released by the Los Alamos National Laboratory in 2010. By applying the pragmatic archiving method, we decreased storage space and time-to-publish by orders of magnitude, as shown in Table 7.1. Simultaneously, the expressiveness of a search operation evolved from DBpedia URI dereferencing only to support for triple pattern queries, including dereferencing.

	first generation	**second generation**
indexing	custom	HDT-CPP
indexing time	~ 24 hours per version	~ 4 hours per version
storage	MongoDB	HDT binary files
space	383 GB	179 GB
# versions	10 versions: 2.0 through 3.9	14 versions: 2.0 through 2015-10
# triples	~ 3 billion	~ 6.2 billion
search expressiveness	subject-based lookup	subject-based lookup triple pattern

Table 7.1: An DBpedia archive based on HDT files decreases storage space and the time-to-publish significantly.

The *first generation* used an archiving approach [14] where DBpedia pages are reconstructed from a custom index, holding the ten DBpedia versions 2.0 to 3.9. Per DBpedia URI, this index stored the RDF description as a blob in MongoDB. Adding a new version triggered a re-indexing of the entire database, resulting in scalability issues that prevented further expansion. With Wikipedia being a constantly changing system, new DBpedia releases are frequent. Hence, a maximum of ten versions is unacceptable for an archive.

We replaced this custom solution with fourteen HDT files for version 2.0 to 2015. The compression ratios are shown in Table 7.2. On average, the size of each HDT file is only 13% of its N-triples counterpart. This has a huge positive impact on the amount of RDF triples that can be archived. In total, the *second generation* archive contains around 6.2 billion triples, which doubles the original amount with less storage use (Table 7.1). Storage space is decreased with 53%, allowing adding more snapshots in the future. The ETL process benefits as well, as it takes twenty hours less on average to put a new DBpedia version into the archive.

As mentioned before, the second generation archive is not restricted to DBpedia pages. It supports efficient look-ups by triple pattern, which facilitates more complex query execution. In the next section, we explain how this capability is leveraged to create a sustainable infrastructure for publishing Linked Data archives on the Web.

Commonly, the *current* version of a dataset approximates the *live* version. As a result, adjusting the generation frequency of new snapshots to the update speed of its live versions can be desirable, ultimately resulting in faster incre-

version	release date	# triples (millions)	N-Triples size (GB)	HDT size (GB)	compression ratio (%)
2.0	July 2007	8.5	2.6	1.4	53.85
3.0	January 2008	120.3	18	2.3	12.78
3.1	July 2008	137	21	2.7	12.86
3.2	October 2008	150	22	2.6	11.82
3.3	May 2009	170	25	2.9	11.60
3.4	September 2009	190	28	3.1	11.07
3.5	March 2010	257	37	4.4	11.89
3.6	October 2010	288	42	4.6	10.95
3.7	July 2011	386	55	5.5	10.00
3.8	July 2012	736	103	6.6	6.41
3.9	September 2013	812	115	7.2	6.26
2014	September 2014	866	123	8.2	6.67
2015-04	April 2015	1,030	142	8.8	6.20
2015-10	October 2015	1,087	149	9.2	6.17

Table 7.2: Fourteen DBpedia versions 2.0 to 2015-10 can be stored with a high average compression rate of 13%.

mental archives. Despite the HDT-based archive serves many archiving scenarios, its update frequency is rather limited. For realistic datasets sizes, the average HDT generation time is expressed in matter of hours. Therefore, scenarios with live updates, i.e., an update frequency below a couple of seconds, require RDF archives that handle small incremental changes better. Research in this domain is still preliminary, yet recent efforts on benchmarking RDF archives [15, 16], strategies [17, 18], change detection in Linked Data [19], and versioned RDF indexing [20, 21], show an increasing interest.

7.5 Publishing versioned Linked Data

Exposing archived RDF versions on the public Web unlocks their real potential. The solutions described in Section 7.4 already add a temporal dimension to each Linked Dataset. Hence, the current version, and each of its archive versions, can be published in a sustainable way through a Triple Pattern Fragments API. Although each individual version is thereby queryable, the relation between them is not. Without prior knowledge, clients cannot automatically navigate from one version to another using the temporal dimension. Fortunately, the Memento [22] framework can enable such by adding time-based versioning to the HTTP resources a Linked Data interface exposes.

7.5.1 The Memento framework

Memento [22] is an HTTP-based framework standardized by IETF, in order to establish a tighter integration between the current and the past Web. In accordance to the REST principles, it enables accessing priorly archived representations of Web resources using common “follow your nose” patterns. It introduces an extra dimension for *content negotiation* between client and server: the *datetime* dimension.

The Memento protocol defines four resource types, their characteristics and the relation between them:

Original Resource URI-R: an existing resource on the Web, of which a user agent is interested in retrieving a past representation from;

Mementos URI-M$_X$ (x=1..n): distinct resources that encapsulate prior states of URI-R at time t_x—if they exist;

Timegate URI-G: a resource supporting content negotiation in the datetime dimension. When requested, it decides on a best matching URI-M$_S$ based on the preferred datetime t_i provided by the client—expressed using the `Accept-Datetime` header;

TimeMap URI-TM (optional): a resource that provides a version history of the URI-R it is associated with, including the URI and version/archival datetime of each known Memento. A user agent finds its way to URI-TM by following a link—expressed in the Link header—provided by a TimeGate, a Memento, or an Original Resource.

Its distinction from the mementos allows URI-R to keep a stable URI, since only old versions get a new one. URI-R is preferably also disconnected from URI-M$_I$ by URI-G to improve caching, instead of datetime negotiated directly. All three resources are connected using hypermedia, as shown in Figure 7.4, which enables navigating between them. We demonstrate this with a brief client-server interaction below.

1. A client discovers URI-G by requesting URI-R and following the link in the HTTP Link header of type `timegate`, which is included in the response.
2. The client requests a past representation from URI-G with *datetime* negotiation. The request contains a `Accept-Datetime:` t_i header, refer-

ring to the desired state valid at that time. Then, the server decides on the closest matching Memento URI-M$_S$ with creation time t_s, with $t_s = min(\forall\ |t_i - t_x|)$. The selected URI-M$_S$ is returned in the `Location` header and with response code `302 Found`, which redirects the client.

3. By following the redirect to URI-M$_S$, the server returns the prior state of URI-R. The response contains an HTTP Link header of type `original` pointing back at URI-R for discoverability. Additionally, links to the `first-memento`, `next-memento` and `prev-memento` relationships can be added to the headers as well.

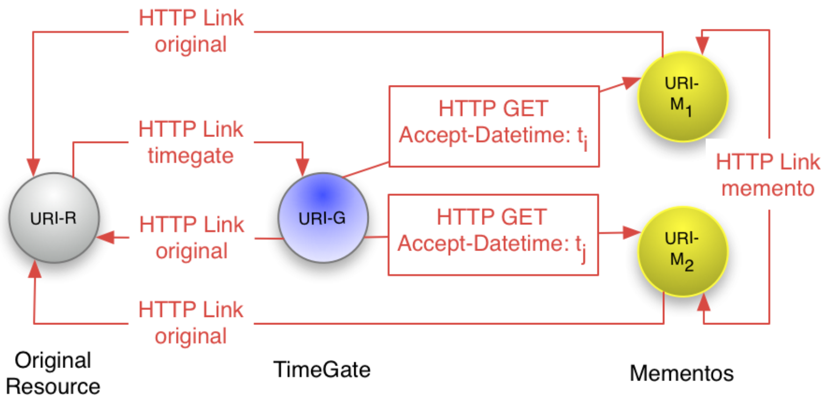


Figure 7.4: Memento adds a time dimension to an Original (Web) Resource URI-R by defining "follow your nose" (hypermedia) patterns between TimeGate URI-G and Memento resources URI-M$_X$. Source: http://www.mementoweb.org/guide/quick-intro

Only the recommended Memento setup is illustrated here; the specification mentions alternative interaction models (e.g., coinciding Original Resource and Timegate) if required due to practical limitations.

Most public web archives around the world[1], including the massive Internet Archive, currently support the Memento protocol. As exemplified by the recent adoption by the W3C[2], there is a growing interest in supporting Memento for resource versioning systems such as wikis, document management systems, and software version control systems. The applicability and power of the Memento protocol for Linked Data has been pointed out as soon as the protocol was initially introduced [14] and since then various efforts have leveraged it [23, 24, 20]. However, these works do not consider a client-side query execution scenario over multiple sources, which is contributed by this chapter.

1. http://mementoweb.org/depot/
2. https://www.w3.org/blog/2016/08/memento-at-the-w3c/

7.5.2 Access to archived Triple Pattern Fragments with Memento

Adding Memento to downloadable data dumps, Linked Data documents or SPARQL endpoints is hindered by their granularity or specification. Downloadable data dumps do not allow modifications to the HTTP layer, preventing to add extra headers. Linked data documents can support Memento, but, as mentioned before, are inadequately expressive for complex query execution. A SPARQL endpoint can answer such queries, but makes versioning hard due to the fine granularity of Web resources it exposes. As mentioned in the previous section, triple indexes that combine SPARQL with versioning are complex and increase server cost even more.

Again, the Triple Pattern Fragments trade-off in interface granularity is beneficial. The amount of Web resources induced by triple patterns is finite and can be generated with minimal cost. This facilitates a smooth integration of the Memento protocol, in addition to low server-side cost and being sufficiently expressive for complex query execution on the client-side.

Supporting Memento implies implementing the three resource types: Original Resource, TimeGate, and Memento. As shown in Figure 7.5, we extend the API to expose the following specific resource types:

TPF Resource: exposes the *current* version (e.g., the most recent HDT file) and is appointed as URI-R.

LDF TimeGate: a resource independent of the interface, as it can be employed for any LDF resource.

TPF Memento: a resource which encapsulates an *archive* version, i.e., exposing a former representation of the TPF resource.

When requesting a TPF, a `Link` header to its corresponding LDF TimeGate is now included in the response. An LDF TimeGate accepts the `Accept-Datetime:` $t \in [t_i, t_j[$ header, which redirects the client to a TPF Memento valid at datetime t. Assuming we use the pragmatic HDT-based RDF archive from Section 7.4, this memento corresponds with an HDT file $H_{[t_i,t_j[}$ retrieved from the archive's index based on t. Once redirected, a client can access $H_{[t_i,t_j[}$ through the TPF Memento $URI_{[t_i,t_j[}$ it was navigated to.

7.5.3 Transparent time-based Web access to the DBpedia archive

We applied this Memento-enabled TPF interface to the DBpedia archive described in Section 7.4.2 to enable live querying historical versions of DBpedia on the public Web. This is established by two physically separate instances. The public DBpedia fragments interface on http://fragments.dbpedia.org provides access to the current DBpedia version and is the official entry point for query-

Support for Memento and the HDT-based archive was implemented in the TPF NodeJS server, available at https://github.com/LinkedDataFragments/Server.js.

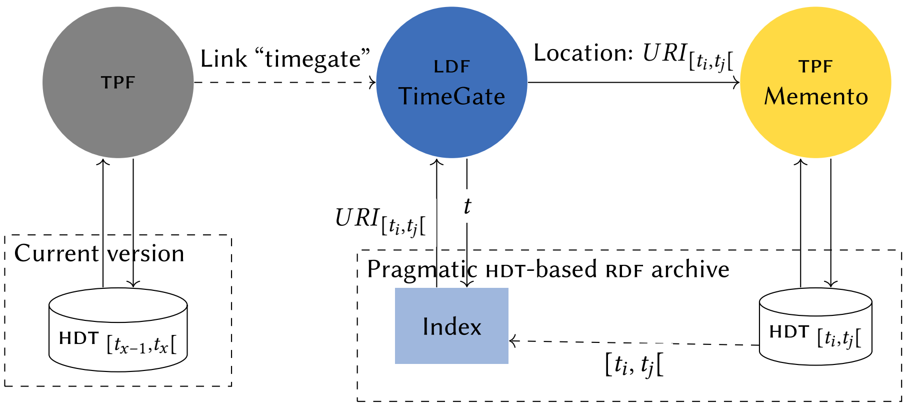


Figure 7.5: A client can transparently navigate from a TPF Resource to a TPF Memento at a specific datetime t.

ing clients. The archived versions of DBpedia reside on http://fragments.mementodepot.org, exposing each of the fourteen HDT files as TPF Memento resources. A `Link` header in each response from http://fragments.dbpedia.org to the DBpedia LDF TimeGate hosted at http://fragments.mementodepot.org/timegate/dbpedia connects both interfaces transparently to clients.

The HTTP communication to negotiate the TPF for `?work dbo:author dbr:Umberto_Eco.` on December 1st, 2013 is illustrated in Listing 7.1.

7.6 Querying versioned and distributed Linked Data

In a virtual integration scenario, the consumer is responsible for physically integrating data from different publishers. Having discussed the technologies and architecture for archiving and versioning above, we now present a use case in which a query is evaluated over a federation of multiple data interfaces on the Web at several points of time in the past.

We first reprise the architecture from Figure 5.1 to add support for versioning over multiple data sources to the TPF client. As already pointed out, a SPARQL query exceeds the expressiveness of a single triple pattern, which forces clients of the TPF interface to break down that query into multiple triple patterns. Therefore, we introduced a Query Engine layer in Section 4.2 that implements a query algorithm which splits a query into one or more TPF requests and produce results by combining the TPF responses locally. These requests are ultimately performed by the HTTP layer, which we have modified to use the Memento protocol, as shown in Figure 7.6a. The client can now choose different points in the past to evaluate the query.

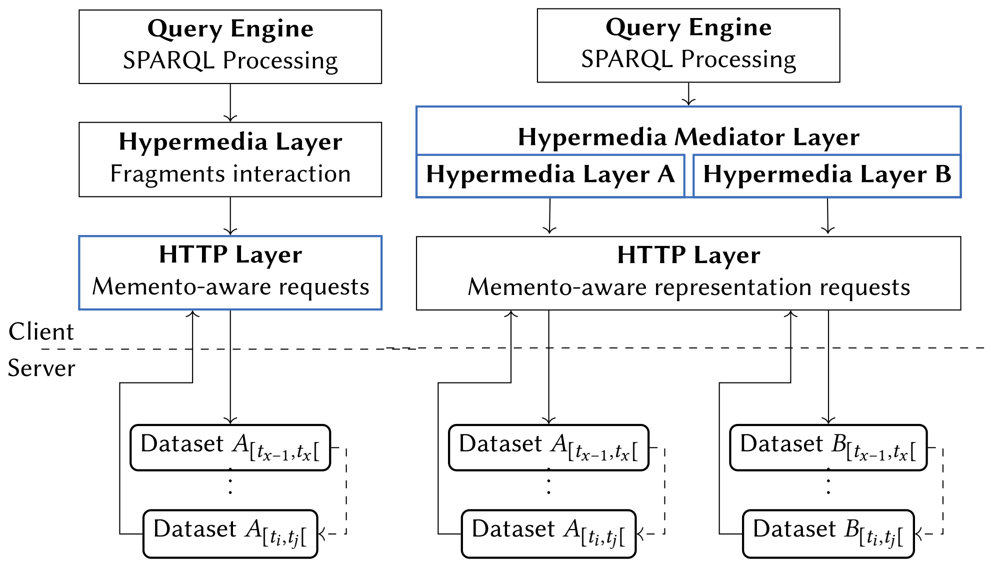
```
(1)
HEAD http://fragments.dbpedia.org/en?subject=&predicate=dbpedia-owl%3Aauthor&
     object=dbpedia%3AUmberto_Eco HTTP/1.1
-----------------------------------------------------------------
HTTP/1.1 200 OK
Link: <http://fragments.mementodepot.org/timegate/dbpedia?subject=&predicate=
     dbpedia-owl%3Aauthor&object=dbpedia%3AUmberto_Eco>;rel=timegate

(2)
GET http://fragments.mementodepot.org/timegate/dbpedia?subject=&predicate=dbpedia-
     owl%3Aauthor&object=dbpedia%3AUmberto_Eco HTTP/1.1
Accept-Datetime: Sun, 01 Dec 2013 22:30:00 GMT
-----------------------------------------------------------------
HTTP/1.1 302 Found
Location: http://fragments.mementodepot.org/dbpedia_3_9?subject=&predicate=dbpedia
     -owl%3Aauthor&object=dbpedia%3AUmberto_Eco

(3)
GET http://fragments.mementodepot.org/dbpedia_3_9?subject=&predicate=dbpedia-owl%3
     Aauthor&object=dbpedia%3AUmberto_Eco HTTP/1.1
-----------------------------------------------------------------
HTTP/1.1 200 OK
Memento-Datetime: Sat, 15 Jun 2013 00:00:00 GMT # = Best matching memento
Link: <http://fragments.dbpedia.org/en?subject=&predicate=dbpedia-owl%3Aauthor&
     object=dbpedia%3AUmberto_Eco>;rel=original,
<http://fragments.mementodepot.org/timegate/dbpedia?subject=&predicate=dbpedia-owl
     %3Aauthor&object=dbpedia%3AUmberto_Eco>;rel=timegate

# Payload
...
```

Listing 7.1: Selects the Memento valid on December 1st, 2013 at 22:30:00 GMT, which is the Memento from June 15, 2013

In combination with the *mediator* extension from Section 5.3, we can consult *multiple* TPF interfaces over time without increasing the complexity of the query engine itself. The definitive client is shown in Figure 7.6b.

Next, as a use case, we focus our attention on the queries formulated in Section 7.3. First, we need to translate each query into SPARQL. Note that we need to include a `UNION` statement to deal with the changing schema, an issue discussed in the next section. The SPARQL adaptation of the single-source query for the number of awards by Belgian academics is shown in Listing 7.2. Running it for different times in the past results in the progression displayed in Figure 7.7.

The SPARQL adaptation of multi-source query for details about professor Jacques-Joseph Haus is shown in Listing 7.3. We evaluated this query at 9 timestamps in the past over UGentMemorialis, VIAF, and DBpedia. Note how every year has a different number of results.



(a) single server with versioning support **(b)** multiple servers with versioning support

Figure 7.6: The client was extended to support versioning and multiple servers, without changing the core query engine.

```
SELECT DISTINCT ?award
WHERE {
  ?person dcterms:subject <http://dbpedia.org/resource/Category:Belgian_academics
       >.
  {
    ?person <http://dbpedia.org/ontology/award> ?award.
  } UNION {
    ?person <http://dbpedia.org/property/awards> ?award.
  }
}
```

Listing 7.2: The SPARQL query to answer "What is the number of awards won by Belgian academics?"

```
SELECT ?professor ?property ?value
WHERE {
  ?professor dbpedia-owl:viafId ?viafId.
  ?professor foaf:name "Haus, Jacques-Joseph".
  ?viafId schema:sameAs ?dbpediaId.
  ?dbpediaId ?property ?value.
}
```

Listing 7.3: The SPARQL query to answer "What is the number of DBpedia triples describing professor 'Haus, Jacques-Joseph' of Ghent University?"

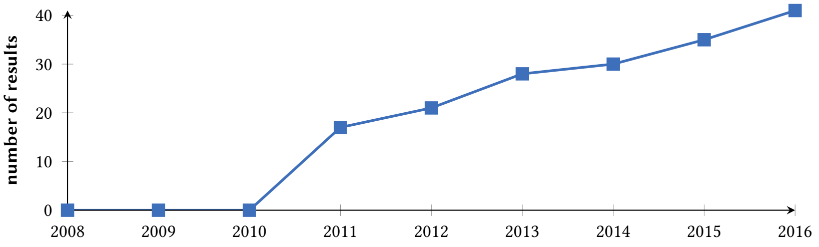


Figure 7.7: By running the same query over DBpedia from 2008 to 2016, we can detect an increase in awards won by Belgian academics.

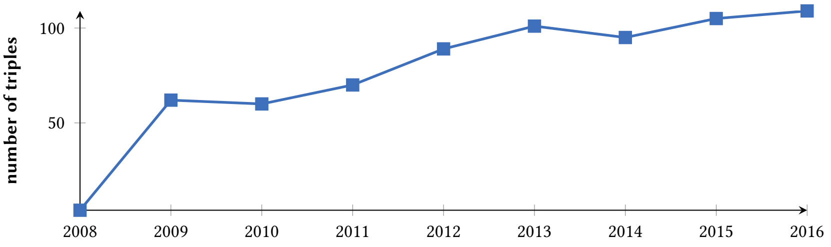


Figure 7.8: By running the same query over UGentMemorialis, VIAF and DBpedia for every year, we gain insight in the evolution of Wikipedia activity for professor Jacques-Joseph Haus.

7.7 Making sense of changing statements

From a technical perspective, executing a single SPARQL query at two different points in time and comparing the result sets, can show the results' evolution. This does not allow, however, to correctly interpret a change between two versions of the same dataset. An RDF statement could have evolved for several reasons:

- it was **added**, because it was not known before;
- it was **known before**, but it was missing; or
- it **replaces** a previously present, but incorrect fact.

In some cases, you could make a few assumptions. For instance, in Figure 7.7 and Figure 7.8, the semantics of an *amount* are hardly debatable. However, one could argue that simply having less triples already misrepresents the facts. The same goes for a changed date when the subject and predicate remain stable; the date might simply be incorrect. Hence, without additional knowledge, semantics can never be deduced with full certainty.

The issues above relate to what Kimball and Ross [25] describe as *slowly changing dimensions* of data warehouses. The assumption that dimension tables are static, is generally considered unrealistic and thus unacceptable. For every attribute within a table, you need a strategy to handle change, which Kimball and Ross categorize as seven basic/hybrid slowly changing dimensions techniques or types. Each type defines how facts should be associated with their stored value.

Type 0: the attribute value did not change; facts remain associated with the original value.
Type 1: the attribute value is overwritten; facts are only associated with the current value (no change tracking).
Type 2: the new attribute value is added to a new duplicate row and the original row is labeled as expired; facts are associated with the value in the then current row.
Type 3: the prior value is added as a new column and the attribute value is overwritten; facts are associated jointly with both the current and prior values.
Type 4: rapidly changing attributes are added in an separate, related mini-dimension table; facts are associated with the then current value in the mini-dimension table.

From the basic types above, three additional hybrid types are composed. Type 5 applies Type 4 while also using Type 1 to kept the current value up to date. Type 6 stores attributes overwritten by Type 1 in a Type 2 row, and all prior rows are also overwritten. Type 7 applies Type 2, but maintains a view limited to the most current rows or attibute values.

A recommendation to discover and retrieve a PROV record is included in the PROV-AQ note[a]. A Link header pointing to the PROV description is added to an HTTP response, linking the resource to its associated record. To also add a time dimension, Coppens et al. [26] proposed a similar principle extended with Memento support.

a. http://www.w3.org/TR/2013/NOTE-prov-aq-20130430

To make sense of changing Linked Data statements, servers and clients will need to apply similar strategies, albeit not only expressed in rows and columns, but also with consideration for semantics and RDF's hypergraph structure. This underlines the importance of making *provenance* information accessible to clients, in order to explain the changes made between versions, i.e., where the current resource state originates from is formally described. This aligns with scientific practice, where publishing results is "being associated with provenance to aid interpretation and trust, and description of methods to support reproducibility" [27].

In Linked Data, data provenance has been a hot research topic for years to establish *data trustworthiness* [28, 29] on the Web. For that reason, the W3C Provenance Working Group released the aforementioned PROV [30] standard, so this information can be uniformly modeled and published. For instance, Listing 7.4 shows an example provenance record in RDF using the PROV-O ontology, which describes the relation between three different versions in the DBpedia archive.

Provenance plays an important role in a *read-write* Web of Linked Data because of its use for trust assessment [31]. This next step enables both human

```
@prefix prov:  <http://www.w3.org/ns/prov#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

<http://fragments.mementodepot.org/dbpedia_2015> prov:wasRevisionOf <http://
      fragments.mementodepot.org/dbpedia_2014>;
    prov:wasGeneratedBy :rev3.

:rev3 prov:endedAtTime "2012-04-18T14:30:00Z"^^xsd:dateTime.

<http://fragments.mementodepot.org/dbpedia_2014> prov:wasRevisionOf <http://
      fragments.mementodepot.org/dbpedia_3_9>;
           prov:wasGeneratedBy :rev2.

:rev2 prov:endedAtTime "2012-04-15T14:30:00Z"^^xsd:dateTime.

<http://fragments.mementodepot.org/dbpedia_3_9> prov:wasRevisionOf <http://
      fragments.mementodepot.org/dbpedia_3_8>;
          prov:wasGeneratedBy :rev1.

:rev1 prov:endedAtTime "2012-04-11T12:30:00Z"^^xsd:dateTime.
```

Listing 7.4: Provenance Record for http://example.org/Resource in PROV-O

and machine users to manipulate Linked Data in addition to querying. Tracking provenance on different levels of granularity will be necessary to explain all these changes. A loyal *sidekick* to PROV is WebID [32]: an open authentication mechanism to identify any agent on the Web by a URI. It has been applied to build distributed social media applications [33] and write decentralized scholarly publications [34, 35]. There is no doubt this combination will continue to gain importance if we manage to scale the Web of Linked Data. Preliminary support for WebID and provenance has been added to the NodeJS implementation of the LDF server[3] [4]

Provenance management is a challenging task. Part of the solution can therefore be generating it on an operation level. Versioning systems already gather important information about changes. Describing this directly as provenance reduces dependency on descriptions by agents or persons [36].

Finally, formalized provenance information can be used to automate decision logic in software systems. For instance, we can implement the Timegate (of the Memento framework) in a loosely coupled and generic way, by applying a semantic reasoner (e.g., the N3 rule reasoner EYE [37]) to the provenance information.

3. https://github.com/LinkedDataFragments/Server.js/wiki/WebID-authentication
4. https://github.com/LinkedDataFragments/Server.js/tree/feature-prov

7.8 Conclusion

This chapter addressed the issue of *reproducibility* when querying distributed Linked Data sources. As Web content drifts over time, query results need to remain valid when they are regenerated later. In that respect, Libraries, Archives and Museums (LAM), in particular the increasing number of under-resourced institutions, struggle to provide a single queryable point of access to their digital preservation efforts. Hence, this chapter proposed an alternative publishing strategy that lowers maintenance cost for publishers, and covers drifting sources (Research Question 5). This entails a shift to a *virtual integration* approach to consolidate these *silos of the LAMs*, i.e., composing a consumer view over distributed datasets that remain in control of the institutions. These claims are supported by a memory reconstruction Use Case based on three complimentary, but distributed Linked datasets (DBpedia, VIAF, and UGentMemorialis).

Considering that most institutions update a "live" RDF database continuously through an Extract-Transform-Load process, publishing recurrently extracted snapshots through a low-cost wrapper stack, avoids the high maintenance cost of exposing the "live" database directly. For cases where snapshots are created at a medium pace (~ daily), the Header-Dictionary-Triples (HDT) format is an excellent storage candidate, as it creates very compact immutable files that are queryable. For DBpedia, this implies an average size gain of 87% compared to the N-Triples format. In combination with the file-system, HDT can serve as pragmatic, but extremely useful RDF archive, which serves basic digital preservation needs with little resources. Cases with higher update frequencies though, i.e., more than once per hour, are momentarily dependent on the ongoing research in RDF archiving systems.

Publishing these snapshots with the Triple Pattern Fragments (TPF) interface in combination with Memento, the HTTP datetime negotiation framework, enables complex queries over current and archived versions of a dataset. Hence, we can validate Hypothesis 5. Potentially, we can gain insight on the evolution of facts in Linked Data by executing the same query at different comparing time points. Without essential provenance metadata about the modifications between versions, however, such insights are not yet reliable. Yet, the same functionality can be used to synchronize distributed sources on the Web, when querying multiple interfaces at once.

This chapter was partly based on the publications:

Miel Vander Sande, Pieter Colpaert, Tom De Nies, Erik Mannens, and Rik Van de Walle. “Publish data as time consistent web API with provenance.” In: *The 23rd International Conference on World Wide Web*. ACM, 2014, pp. 953–958

Miel Vander Sande, Ruben Verborgh, Patrick Hochstenbach, and Herbert Van de Sompel. “Towards sustainable publishing and querying of distributed Linked Data archives.” In: *Journal of Documentation* (2017)

References

[1] Holm Tetens. “Reproducibility, Objectivity, Invariance.” In: *Reproducibility: Principles, Problems, Practices, and Prospects* (2016).

[2] Leo Sauermann, Richard Cyganiak, and Max Völkel. *Cool URIs for the Semantic Web*. Saarländische Universitäts-und Landesbibliothek, 2011.

[3] Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres, and Stefan Decker. “Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources.” In: *The WWW2010 Workshop on Linked Data on the Web*. Apr. 27, 2010. URL: http://ceur-ws.org/Vol-628/ldow2010%5C_paper12.pdf.

[4] Amanda Kay Rinehart, Patrice-Andre Prudhomme, and Andrew Reid Huot. “Overwhelmed to action: digital preservation challenges at the under-resourced institution.” In: *OCLC Systems & Services* 30.1 (2014), pp. 28–42.

[5] Margaret L Hedstrom and Sheon Montgomery. *Digital preservation needs and requirements in RLG member institutions*. Research Libraries Group Mountain View, Calif., 1998.

[6] Edward M Corrado and Heather Lea Moulaison. *Digital preservation for libraries, archives, and museums*. Rowman & Littlefield, 2014.

[7] Karen Smith Yoshimura. “Analysis of International Linked Data Survey for Implementers.” In: *D-Lib Magazine* 22.7 (2016), p. 6.

[8] Françoise Bourdon and Vincent Boulet. “VIAF: A hub for a multilingual access to varied collections.” In: *World library and information Congres: 78th IFLA general Conference and Assembly*. 2013.

[9] Christophe Verbruggen and Gita Deneckere. *UGentMemorialis. Biographical data of UGent professors between 1817 and 2012 [Dataset]*. Ed. by Christophe Verbruggen and Gita Deneckere. 2016. URL: http://www.UGentMemorialis.be.

[10] George Papastefanatos. “Challenges and Opportunities in the Evolving Data web.” In: *International Conference on Conceptual Modeling*. Springer, 2013, pp. 23–28.

[11] Bernadette Houghton. "Preservation Challenges in the Digital Age." In: *D-Lib Magazine* 22.7 (2016), p. 1. URL: http://dlib.org/dlib/july16/houghton/07houghton.html.

[12] Seamus Ross. "Digital preservation, archival science and methodological foundations for digital libraries." In: *New Review of Information Networking* 17.1 (2012), pp. 43–68.

[13] Jaime Schumacher, Lynne M. Thomas, Drew Vande Creek, Stacey Erdman, Jeff Hancks, Aaisha Haykal, Meg Miner, Patrice-Andre Prudhomme, and Danielle Spalenka. "From Theory to Action:'Good Enough' Digital Preservation Solutions for Under-Resourced Cultural Heritage Institutions." In: *Institute of Museum and Library Services* (2014).

[14] Herbert Van de Sompel, Robert Sanderson, Michael L. Nelson, Lyudmila Balakireva, Harihar Shankar, and Scott Ainsworth. "An HTTP-Based Versioning Mechanism for Linked Data." In: *CoRR* abs/1003.3661 (2010). URL: http://arxiv.org/abs/1003.3661.

[15] Javier D Fernández, Jürgen Umbrich, Axel Polleres, and Magnus Knuth. "Evaluating query and storage strategies for RDF archives." In: *The 12th International Conference on Semantic Systems.* ACM, 2016, pp. 41–48.

[16] Marios Meimaris and George Papastefanatos. "The EvoGen Benchmark Suite for Evolving RDF Data." In: *MEPDaW/LDQ@ ESWC.* 2016, pp. 20–35.

[17] Marvin Frommhold, Rubén Navarro Piris, Natanael Arndt, Sebastian Tramp, Niklas Petersen, and Michael Martin. "Towards versioning of arbitrary RDF data." In: *The 12th International Conference on Semantic Systems.* ACM. 2016, pp. 33–40.

[18] Vassilis Papakonstantinou, Giorgos Flouris, Irini Fundulaki, Kostas Stefanidis, and Giannis Roussakis. "Versioning for Linked Data: Archiving Systems and Benchmarks." In: *BLINK@ ISWC.* 2016.

[19] Renata Dividino, André Kramer, and Thomas Gottron. "An Investigation of HTTP Header Information for Detecting Changes of Linked Open Data Sources." In: *The Semantic Web: ESWC 2014 Satellite Events: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers.* Ed. by Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai. Cham: Springer International Publishing, 2014, pp. 199–203. ISBN: 978-3-319-11955-7. DOI: 10.1007/978-3-319-11955-7_18.

[20] Paul Meinhardt, Magnus Knuth, and Harald Sack. "TailR: A Platform for Preserving History on the Web of Data." In: *The 11th International Conference on Semantic Systems.* SEMANTICS '15. Vienna, Austria: ACM, 2015, pp. 57–64. ISBN: 978-1-4503-3462-4. DOI: 10.1145/2814864.2814875.

[21] Javier D Fernández, Axel Polleres, and Jürgen Umbrich. "Towards Efficient Archiving of Dynamic Linked Open Data." In: *DIACRON@ ESWC.* 2015, pp. 34–49.

[22] Herbert Van de Sompel, Michael Nelson, and Robert Sanderson. *HTTP Framework for Time-Based Access to Resource States – Memento.* Request For Comments 7089. Internet Engineering Task Force, Dec. 2013. URL: https://tools.ietf.org/rfc/rfc7089.

[23] Javier D. Fernández, Patrik Schneider, and Jürgen Umbrich. "The DBpedia Wayback Machine." In: *The 11th International Conference on Semantic Systems.* SEMANTICS '15. Vienna, Austria: ACM, 2015, pp. 192–195. ISBN: 978-1-4503-3462-4. DOI: 10.1145/2814864.2814889.

[24] Miel Vander Sande, Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Adding time to Linked Data: a generic Memento proxy through PROV." In: *Poster and Demo of the 12th International Semantic Web Conference.* Sydney, Australia: CEUR-WS.org, 2013, pp. 217–220. ISBN: 9783642413346. URL: http://www.iswc2013.semanticweb.org/sites/default/files/iswc%5C%5C_poster%5C%5C_10.pdf.

[25] Ralph Kimball and Margy Ross. *The data warehouse toolkit: The definitive guide to dimensional modeling.* John Wiley & Sons, 2013.

[26] Sam Coppens, Erik Mannens, Davy Van Deursen, Patrick Hochstenbach, Bart Janssens, and Rik Van de Walle. "Publishing provenance information on the web using the memento datetime content negotiation." In: *WWW2011 workshop on Linked Data on the Web (LDOW 2011).* Vol. 813. 2011, pp. 6–15.

[27] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, et al. "Why linked data is not enough for scientists." In: *Future Generation Computer Systems* 29.2 (2013), pp. 599–611.

[28] Davide Ceolin, Paul Groth, Willem Robert Van Hage, Archana Nottamkandath, and Wan Fokkink. "Trust evaluation through user reputation and provenance analysis." In: *The 8th International Conference on Uncertainty Reasoning for the Semantic Web-Volume 900.* CEUR-WS.org. 2012, pp. 15–26.

[29] Elisa Bertino. "Data Trustworthiness—Approaches and Research Challenges." In: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance.* Springer, 2015, pp. 17–25.

[30] Yolanda Gil and Simon Miles. "PROV Model Primer—W3C Working Group Note." In: *World Wide Web Consortium (W3C)* (2013).

[31] Sam Coppens, Ruben Verborgh, Miel Vander Sande, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. "A truly Read-Write Web for machines as the next-generation Web?" In: *The SW2022 workshop: What will the Semantic Web look like 10 years from now?* Nov. 2012. URL: http://stko.geog.ucsb.edu/sw2022/sw2022%5C_paper3.pdf.

[32] Manu Sporny, Toby Inkster, Henry Story, Bruno Harbulot, and Reto Bachmann-Gmür. *WebID 1.0: Web identification and discovery.* Editor's draft. World Wide Web Consortium, 2011.

[33] Andrei Sambra, Amy Guy, Sarven Capadisli, and Nicola Greco. "Building Decentralized Applications for the Social Web." In: *The 25th International Conference Companion on World Wide Web*. WWW '16 Companion. Montreal, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 1033–1034. ISBN: 978-1-4503-4144-8. DOI: 10.1145/2872518.2891060.

[34] Matthew Gamble and Carole Goble. "Standing on the shoulders of the trusted Web: Trust, Scholarship and Linked Data." In: *The WebSci10: Extending the Frontiers of Society*. 2010.

[35] Sarven Capadisli, Amy Guy, Ruben Verborgh, Christoph Lange, Sören Auer, and Tim Berners-Lee. "Decentralised Authoring, Annotations and Notifications for a Read-Write Web with dokieli." In: *Web Engineering: 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*. Ed. by Jordi Cabot, Roberto De Virgilio, and Riccardo Torlone. Cham: Springer International Publishing, 2017, pp. 469–481. ISBN: 978-3-319-60131-1. DOI: 10.1007/978-3-319-60131-1_33.

[36] Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. "R&Wbase: Git for Triples." In: *Linked Data On the Web Workshop*. 2013.

[37] Jos De Roo. *Euler Proof Mechanism*. 2013. URL: http://eulersharp.sourceforge.net.

We can't fix your heart, but thanks to modern technology we can tell you exactly how damaged it is!

— Dr Hibbert

Conclusions 8

In this dissertation, I have investigated if alternative Web interfaces can lead to better Web-scale publication of Linked Data. I discussed the Linked Data Fragments conceptual model to analyze existing Web APIs *to* RDF *data and define new ones. Consequently, I applied this model to introduce the Triple Pattern Fragments interface to offer* SPARQL *querying to clients with minimal server cost. Then, this interface was employed to evaluate the effects of different metadata on complex client-side query execution, execute queries over multiple sources, discover data sources through hypermedia, and effectively publish* RDF *archives. This chapter lists my general findings, revisits the research questions posed in Chapter 1, and sketches a general outlook.*

On a Semantic Web, intelligent software agents are able to autonomously discover, query and understand data. However, if the Web of Linked Data is a practical deployment of the Semantic Web, the Web APIs that exist to publish Linked Data do not suffice. This research was instigated by the low availability of *live* queryable Linked Data. Despite the growing amount of Linked Open Data in existence, a high number of datasets on the public Web is either *(a)* not queryable with SPARQL; or *(b)* subject to frequent downtimes. Considering the cost and complexity of hosting a public SPARQL endpoint, it is no surprise many under-resourced institutions struggle to provide a single queryable point of access to their data and resort to hosting downloadable data dumps.

This dissertation therefore opened the discussion on the complete spectrum of possible Web interfaces, which is embodied by the Linked Data Fragments model. The current lack of alternatives to SPARQL endpoints suggest a focus on large aggregated RDF knowledge bases, publishers with sufficient funds, and a disregard for established characteristics of the public Web. While there is inherently nothing wrong with the concept of a SPARQL endpoint—it is perfectly acceptable in a private, predictable environment—the failure to provide a reliable query service has huge consequences for the development of Semantic

Web applications. This is a problem inherent to the concept of powerful endpoint and, thus, cannot be solved by continuing to improve the efficiency of SPARQL servers.

Instead of creating intelligent servers, the interface should facilitate clients to be intelligent. I therefore defined and evaluated alternative Web interfaces that *democratize* publishing datasets and enable clients to execute complex queries. More attention for smaller datasets from under-resourced data institutions—primarily in the Open Government Data or Cultural Heritage domain—is crucial to unlocking more knowledge, as they are often owners of highly specific and carefully curated data. This aligns with the *visible library* [1] concept: get your data out there, and solutions to other data integration problems (e.g., details about vocabularies and modeling) will follow.

8.1 Review of the Research Questions

The findings in this doctoral work accompanied five research questions I raised in Section 1.1. I first discuss whether the outcomes have provided me with the right answers.

In search for an economical and technological sustainable compromise on a Web interface, I posed my main Research Question 1:

"Can a Web interface for Linked Datasets designed for low server cost enable complex live querying for clients?"

The introduced Triple Pattern Fragments interface (Chapter 4) achieves low server cost by restricting the query granularity to a triple pattern—the basic element of a SPARQL query. On the client-side, a query is split into triple patterns, which results into multiple requests. Compared to a SPARQL endpoint, general CPU load is significantly lower and more stable during query execution, which is due to the coarse granular requests and increased cache hits. Thus, the Triple Pattern Fragments interface achieves low server cost, and by demanding less complex RDF indexes, enables affordable infrastructure for many publishers. Queries do execute slower, but—despite using a greedy query algorithm—stay well within the four-second limit for the real-world queries from the DBpedia SPARQL benchmark [2] and are unaffected by the number of clients. In terms of external validity, the LODLaundromat (http://lodlaundromat.org/wardrobe) indicates that, at the time of the writing, around 658,045 Linked Datasets exist, each of which is available as a Triple Pattern Fragments interface [3]. Consequently, a large number of datasets that could previously only be reliably published as data dumps, can now be provided with live queryable access, which validates Hypothesis 1 *"A client can execute real-world queries over a restricted Linked Data Web interface with a response time below four seconds and a lower server CPU load than more expressive interfaces."* Hence, enabling more nuance and demanding less from servers can ultimately increase productivity with Linked Data.

In exchange for low server cost, we accept high HTTP bandwidth demands and slower execution time. We accept this trade-off as inevitable, yet, as we learn from database literature [4], the client performance can still be improved with metadata. However, the practical aspects of Linked Data querying have been understudied with sole focus on precision and recall, thus we posed Research Question 2:

"To what extent can interface metadata enhance complex query evaluation in a practical Web setting?"

The *cardinality* metadata included in the Triple Pattern Fragments definition can be efficiently computed when using specific indexes such as HDT [5]. With the request of a single page, a client-side query algorithm determines the most selective pattern and optimizes locally what triple pattern needs to be joined first. While this renders acceptable results for many query types, it is problematic for some. Cartesian products and a dominance of membership checks (checking whether a triple is part of the dataset or not) creates substantial HTTP bandwidth overhead. The choice in RDF serialization for the response only has a minimal impact and only requires further efforts for future less *greedy* query algorithms. Additional *approximate membership metadata* significantly reduces the amount of HTTP request for affected query types, but introduces counterproductive overhead per request. Such metadata can only lower execution time when precached and/or provided to the client as a separate resource. We can validate Hypothesis 2 *"Extra metadata reduces at least one third of the network traffic required by clients to answer a query."*; but clearly the discussion is broader than reducing requests, motivating more multi-faceted research on different metadata types. Adding extra interface metadata does show the merits of self-descriptive fragments. Machine processors can discover the capabilities of the query endpoints in an automated way, which they can use or ignore as they see fit. The trade-off between server cost/availability and client performance will continue to exist, but benefits from uniformity. This is opposed to traditional Semantic Web of different endpoints that implement different standards, versions or features.

Despite possible performance improvements, one could argue whether our expectations "ask a complex question; wait for the result" of SPARQL querying are sustainable at Web-scale. Being an open and unpredictable Web, a finite-length response seems counter-intuitive; the same goes for instant query response times. This dissertation therefore re-evaluated how applications are developed on top of Linked Data with *(a)* requiring support for incremental results, i.e., start processing results during query execution, and *(b)* introduce opportunistic querying, i.e., temporarily allow imprecise results.

The upcoming call for a *decentralized Web* [6]—services and applications on the Web no longer depend on a single central organization to function—reinforces this reflection. One of its implications is an extensive distribution of data over many sources, making the retainment of a traditional query paradigm futile, and the quest for innovative paradigms imminent. Accordingly, one could argue if SPARQL is here to stay as query language, or whether it will eventually be replaced by more fitting proposals [7].

The notion of Web-scale thus goes beyond a single data source; Linked Data interconnects datasets to create a global, machine understandable data space. Data publishers are therefore urged to implement a virtual integration strategy to deal with the loosened control, synchronization problems, and high infrastructural costs of centralized dataset aggregation. Instead, a consumer view is composed over distributed datasets that remain in control of the data publishers. Triple Pattern Fragments could be a suitable candidate to deploy virtual integration in a sustainable manner, which provoked Research Question 3:

"Can restricted low-cost Linked Data Web interfaces form an efficient architecture for query evaluation over a federation of interfaces?"

The Triple Pattern Fragments interface natively adopts the client-server paradigm found in SPARQL query federation frameworks (Chapter 5). Even when running on a public network, query completeness is competitive with the state-of-the-art, and even for some queries, the execution time is comparable. Hypothesis 3 *"A client can evaluate queries over a federation of low-cost interfaces on a public network with a performance similar to the state-of-the-art federation frameworks ANAPSID, FedX, and SPLENDID."* can thus be validated, and, considering no source selection or advance query algorithm has been applied, this approach has proven itself very promising for query federation. Rather than seizing server-side control of this costly task, clients are provided with the resources needed to perform federation themselves. Its self-descriptive responses can eventually deal with federations of heterogeneous interfaces through uniformity.

Before any federation can be queried, the interfaces need to be discovered first. This insight led to Research Question 4:

"How can we effectively discover distributed Linked Data interfaces?"

With Linked Data heavily relying on hyperlinks, it seems evident to apply this scale-free HTTP network for this purpose. Hence, I proposed an approach where interfaces can discover each other by exploiting the Linked Data principles to exchange summaries about their dataset. Results showed that the overhead is low, especially because the discovery process happens in the background (Chapter 6). The collected summaries benefit client-side source selection, as the links between datasets can partially predict the interfaces relevant to queries. It would be interesting to see what integration with existing source selection techniques could yield. Using hyperlinks for communicating relevant interfaces to clients requires more intelligent consumption, yet they blur the distinction between query execution and federated query execution, yielding an attractive path towards a global data space. However, the findings were insufficient to validate Hypothesis 4 *"A client can effectively discover distributed interfaces by relying solely on Linked Data principles."*

Finally, the distributed interfaces that result from a virtual integration strategy also drift over time. This creates a problem of reproducibility, where interfaces become out-of-sync and queries can no longer be executed in the same context. However, with many stakeholders already investing in digital preservation, I can answer Research Question 5:

"Can low-cost interfaces improve access to prior versions of Linked Datasets?"

This reproducibility problem can be tackled by also publishing the preservation efforts performed by Linked Data publishers in the same low-cost manner (Chapter 7). The Triple Pattern Fragments interface is a suitable fit to support the Memento protocol—a standardized HTTP framework for time-based resource versioning. In combination with a (pragmatic) RDF archiving system, distributed interfaces can be transparently synchronized around a given date-time during query execution; thus, creating a valid context to reproduce results, which is crucial in order to revive virtual integration. Hence, I conclude with validating Hypothesis 5 *"Clients can query published prior versions of Linked Datasets without specifying the exact version."*

By comparing results from different timepoints, the added time dimension also enables analyzing the evolution of facts. However, without fine-grained provenance information about the changes, we should be careful with the semantics of such evolution. In general, provenance has a bigger role to play in explaining how a client obtained an answer, what data sources were used in the process, and ultimately, how the answer can be trusted.

8.2 Open challenges and future directions

We conclude this thesis by listing the remaining general challenges and possible directions for developing the introduced technologies.

In this thesis, we have made a case for data publishers to adopt a virtual integration strategy and supplied an architecture based on Triple Pattern Fragments, which is already deployable in practice. One of the main challenges though, is to obtain a sufficiently fast arrival of results. Evidently, the definition of "fast" entirely depends on an application's demands in completeness, i.e., do we strive for efficiently retrieving the first result, the last result, half of the results, or all results? We stress that it is inherently difficult to impossible to achieve the same performance as with physical integration; however, the question is rather whether the achieved performance is sufficient to support the envisaged applications or use cases, and if not, how it can be improved. As every compromise sacrifices a benefit on either the consumer side or the publisher side, finding sweet spots requires thorough research and evaluation in context of their application domain. For instance, the Triple Pattern Fragments interface balances between the expressivity of a SPARQL endpoint and the low computational cost of data dumps, at the cost of increased bandwidth and query execution time.

Immediate improvements are possible regarding the current greedy client-side query algorithm. The number of necessary requests can be reduced significantly, both for a single-interface and multiple-interfaces setup. Improvements were already proposed by Van Herwegen et al. [8] and Acosta and Vidal [9]. The former migrates decisions from local to global: based on multiple heuristics, the client estimates from the intermediate results whether the greedy approach is suboptimal. If so, the triple patterns are downloaded separately instead, resulting in fewer requests. However, because the join process is more complex, it requires more computational work from the client. The latter introduced a query engine that better adapts to unexpected data source conditions. A bushy tree query plan reduces intermediate results and the schedule is adaptive to live conditions, such as interface response times. In addition to exploring these existing methods further, many optimization techniques from relational databases [10], distributed databases [11], or RDF databases [12] are directly applicable. However, their effectiveness is determined by the live Web environment and the limited expressiveness of the interface. Literature closest to those circumstances are in the field of federated SPARQL querying [13]. Approaches that significantly impact page size would also make revisiting response serialization worthwhile.

Eventually, improvements on the query algorithm will reach a limit, and research will have to expand beyond Triple Pattern Fragments, i.e., equipping the server-side interface with additional interface features. Examples contained in this thesis were the metadata extension with approximate membership, and the hypermedia controls extension for discovery and versioning. Other work has modified the selector to facilitate textual searches [14] or move certain types of joins from clients to servers [15]. Similar exploration of the Linked Data Fragments axis—with accompanying client-side adjustments—is certainly endorsed, in particular differences between independent and conjunctive use. Of great value is the recent work by [16], which presents a formal framework that enables assessing new Linked Data Fragments interface features before implementation. Among other things, the article shows that the rather conceptual axis used in this dissertation, is in fact better represented as an *expressiveness lattice*. In addition to new interface features, different inclusion techniques should be investigated. For instance, is metadata included in the response, or available through a separate resource? Also, vocabularies that can effectively describe interface features and facilitate the client's utility, need to be studied accordingly.

For federated queries, a more sophisticated mechanism for source selection can eliminate several of them beforehand for (parts of) a given query, reducing the number of needed HTTP requests. Applying this process involves (client-side or server-side) pre-processing, which in the case of an RDF archive (multiple versions per dataset) means that extra processing per version would be required.

Interface discovery can also benefit from source selection and query execution time for use cases which do not need 100% completeness. It is likely that additional hypermedia or metadata repositories are necessary to improve completeness, possibly constructed by a server dedicated to discovery. A direct performance improvement is taking into account practical differences between Linked Data interfaces, e.g., response times or page size, while communicating discovery outcome to the client. Query efficiency can also be improved further by integrating discovery with existing source selection methods. The query algorithm itself can take advantage of the data summaries currently used for discovery. However, it remains to be investigated to what extent this extra effort, if performed by the server, could benefit both the discovery mechanism as for query execution (reduce query times on the client) as such. Finally, source selection techniques in general could benefit from machine learning to model a network of interfaces. Starting from the total set of possible sources, a client could, for instance, apply binary classifiers to construct a model from the incoming fragments, with the eventual relevant sources as positive training data. Or reinforcement learning could be used for discovering a small network of interfaces, with the possible links as actions and a visited interface' relevance as possible reward. Of course, the research challenge would be to extract features from a fragment's data, metadata, or hypermedia, preferably with sufficient performance to train during query execution. This does however bring us closer to a real intelligent client.

Another challenge is *usability*: the more accessible the presented techniques become, the higher the chances of adoption. On the one hand, usability improvements should focus on reducing the time and effort for data publishers to bring their data online, as well as version management. On the other hand, query execution should be facilitated for data consumers. This includes support for writing queries, which can become complicated—especially when multiple sources with different vocabularies are consulted. For added ease of use, more simple query languages should be supported by the client, including visual languages that allow users to create queries through drag and drop.

Another issue related to usability, is the lack of clear schema evolution management for Linked Open Datasets. For instance, the public datasets DBpedia and VIAF have both introduced significant changes to the schema over time, aggravating the reproducibility problem and contributing to the *vocabulary chaos* [17]. In the continuing effort to expose information as Linked Data, Dunsire et al. [17] urge to double our efforts on supporting infrastructure, along with "guiding principles and best practices around reuse, extension of existing vocabularies, as well as development of new vocabularies". Of course, semantic changes are inevitable; however, it is the deprecation processes—ensuring backwards compatibility and communicating version changes—that badly requires clear planning and interoperability. Furthermore, in line with the infrastructure presented in this dissertation, application developers indicate the desire for "machine-readable, API-based access to version history" in vocabulary registries as well [17].

While the list of challenges remains long, I hope this thesis provided the means to invite more stakeholders to the Web of Linked Data, with application development gaining because of that. Although it is tempting as a community to preach to the choir, we must not restrain ourselves from reflecting critically at established practice (e.g., the application of public SPARQL endpoints) and admit that sometimes innovation happens through simplification. Realizing a Semantic *Web*—regardless of its ultimate shape or form—requires more attention to the Web as a running system, its architecture, and the characteristics that made it succeed; especially because it is exactly that what semantic technologies complement to other fields such as Artificial Intelligence and Big Data analytics. Intelligent agents can rise if each field does its part, but also strengthens the ties between them; I know I will go on doing so.

References

[1] Eric Miller and Uche Ogbuji. "Linked data design for the visible library." In: *Bulletin of the Association for Information Science and Technology* 41.4 (2015), pp. 23–29. ISSN: 2373-9223. DOI: 10.1002/bult.2015.1720410409.

[2] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. "DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data." In: *The 9th International Semantic Web Conference.* 2011. ISBN: 978-3-642-25072-9.

[3] Laurens Rietveld, Ruben Verborgh, Wouter Beek, Miel Vander Sande, and Stefan Schlobach. "Linked Data-as-a-Service: The Semantic Web Redeployed." In: *European Semantic Web Conference.* Ed. by Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann. Springer International Publishing, June 2015, pp. 471–487.

[4] Yannis E Ioannidis. "Query optimization." In: *ACM Computing Surveys (CSUR)* 28.1 (1996), pp. 121–123.

[5] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. "Binary RDF Representation for Publication and Exchange (HDT)." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (Mar. 2013), pp. 22–41.

[6] Luis-Daniel Ibáñez, Elena Simperl, Fabien Gandon, and Henry Story. "Redecentralizing the Web with Distributed Ledgers." In: *IEEE Intelligent Systems* 32.1 (Jan. 2017), pp. 92–95. ISSN: 1541-1672. DOI: 10.1109/MIS.2017.18.

[7] Olaf Hartig and Jorge Pérez. "LDQL: A Query Language for the Web of Linked Data." In: *The Semantic Web - ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I.* Ed. by Marcelo Arenas et al. Cham: Springer International Publishing, 2015, pp. 73–91. ISBN: 978-3-319-25007-6. DOI: 10.1007/978-3-319-25007-6_5. URL: https://doi.org/10.1007/978-3-319-25007-6%5C_5.

[8] Joachim Van Herwegen, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Query Execution Optimization for Clients of Triple Pattern Fragments." In: *The 12th Extended Semantic Web Conference*. Ed. by Fabien Gandon, Marta Sabou, Harald Sack, Claudia d'Amato, Philippe Cudré-Mauroux, and Antoine Zimmermann. June 2015.

[9] Maribel Acosta and Maria-Esther Vidal. "Networks of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data." In: *The Semantic Web – ISWC 2015*. Ed. by Marcelo Arenas et al. Vol. 9366. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 111–127. ISBN: 978-3-319-25006-9.

[10] Alon Y Halevy. "Answering queries using views: A survey." In: *The VLDB Journal* 10.4 (2001), pp. 270–294.

[11] Luis Galárraga, Katja Hose, and Ralf Schenkel. "Partout: a distributed engine for efficient RDF processing." In: *The 23rd International Conference on World Wide Web*. ACM, 2014, pp. 267–268.

[12] Michael Schmidt, Michael Meier, and Georg Lausen. "Foundations of SPARQL Query Optimization." In: *The 13th International Conference on Database Theory*. ACM, 2010, pp. 4–33.

[13] Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. "A comparison of federation over SPARQL endpoints frameworks." In: *International Conference on Knowledge Engineering and the Semantic Web*. Springer. 2013, pp. 132–146.

[14] Joachim Van Herwegen, Laurens De Vocht, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Substring Filtering for Low-Cost Linked Data Interfaces." In: *The 14th International Semantic Web Conference*. Ed. by Marcelo Arenas et al. Oct. 2015.

[15] Olaf Hartig and Carlos Buil-Aranda. "Bindings-Restricted Triple Pattern Fragments." In: *The On the Move to Meaningful Internet Systems*. Ed. by Christophe Debruyne, Hervé Panetto, Robert Meersman, Tharam Dillon, eva Kühn, Declan O'Sullivan, and Claudio Agostino Ardagna. Cham: Springer International Publishing, 2016, pp. 762–779. ISBN: 978-3-319-48472-3. DOI: 10.1007/978-3-319-48472-3_48.

[16] Olaf Hartig, Ian Letter, and Jorge Pérez. "A Formal Framework for Comparing Linked Data Fragments." In: *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*. Ed. by Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin. Cham: Springer International Publishing, 2017, pp. 364–382. ISBN: 978-3-319-68288-4. DOI: 10.1007/978-3-319-68288-4_22.

[17] Gordon Dunsire, Corey Harper, Diane Hillmann, and Jon Phipps. "Linked data vocabulary management: infrastructure support, data integration, and interoperability." In: *Information Standards Quarterly* 24.2/3 (2012), pp. 4–13.