Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)
genehmigte Dissertation von
Dipl.-Wi.-Ing. Tobias Christof Käfer

Behaviour on Linked Data

Specification, Monitoring, and Execution

Tobias Christof Käfer

Tag der mündlichen Prüfung: 18. Dezember 2018
Referent: Prof. Dr. York Sure-Vetter
Korreferent: Prof. Dr. Andreas Harth
Korreferent: Prof. Dr. Axel Polleres
Karlsruhe, 2019

Abstract

People, organisations, and machines around the globe make use of web technologies to communicate. For instance, 4.16 bn people with access to the internet made 4.6 bn pages on the web accessible using the transfer protocol HTTP, organisations such as Amazon built ecosystems around the HTTP-based access to their businesses under the headline RESTful APIs, and the Linking Open Data movement has put billions of facts on the web available in the data model RDF via HTTP. Moreover, under the headline Web of Things, people use RDF and HTTP to access sensors and actuators on the Internet of Things.

The necessary communication requires interoperable systems at a truly global scale, for which web technologies provide the necessary standards regarding the transfer and the representation of data: the HTTP protocol specifies how to transfer messages, besides defining the semantics of sending/receiving different types of messages, and the RDF family of languages specifies how to represent the data in the messages, besides providing means to elaborate the semantics of the data in the messages. The combination of HTTP and RDF –together with the shared assumption of HTTP and RDF to use URIs as identifiers– is called Linked Data.

While the representation of static data in the context of Linked Data has been formally grounded in mathematical logic, a formal treatment of dynamics and behaviour on Linked Data is largely missing. We regard behaviour in this context as the way in which a system (e. g. a user agent or server) works, and this behaviour manifests itself in dynamic data. Using a formal treatment of behaviour on Linked Data, we could specify applications that use or provide Linked Data in a way that allows for formal analysis (e. g. expressivity, validation, verification). Using an experimental treatment of behaviour, or a treatment of the behaviour's manifestation in dynamic data, we could better design the handling of Linked Data in applications.

Hence, in this thesis, we investigate the notion of behaviour in the context of Linked Data. Specifically, we investigate the research question of how to capture the dynamics of Linked Data to inform the design of applications. The first contribution is a corpus that we built and analysed to monitor dynamic Linked Data on the web to study the update behaviour. We provide an extensive analysis to set up a long-term study of the dynamics of Linked Data on the web. We analyse data from the long-term study for dynamics on the level of accessing changing documents and on the level of changes within the documents. The second contribution is a model of computation for Linked Data that allows for expressing executable specifications of application behaviour. We provide a mapping from the conceptual foundations of the standards around Linked Data to Abstract State Machines, a Turing-complete model of computation rooted in mathematical logic. The third contribution is a workflow ontology and corresponding operational semantics to specify applications that execute and monitor behaviour in the context of Linked Data. Our approach allows for monitoring and executing behaviour specified in workflow models and respects the assumptions of the standards and practices around Linked Data. We evaluate our findings using the experimental corpus of dynamic Linked Data on the web and a synthetic benchmark from the Internet of Things, specifically the domain of building automation.

Publications

Parts of this thesis have already been published. Specifically, this thesis is based on the following publications:

In Peer-Reviewed Conference Proceedings:

- Tobias Käfer and Andreas Harth. "Specifying, Monitoring, and Executing Workflows in Linked Data Environments". In: *Proceedings of the 17th International Semantic Web Conference (ISWC)*. 2018, pp. 424–440
- Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne and Aidan Hogan. "Observing Linked Data Dynamics". In: *Proceedings of the 10th European Semantic Web Conference (ESWC)*. 2013, pp. 213–227. Our corresponding poster won the Best Poster Award

In Peer-Reviewed Workshop Proceedings:

- Tobias Käfer and Andreas Harth. "Rule-based Programming of User Agents for Linked Data". In: *Proceedings of the 11th International Workshop on Linked Data on the Web (LDOW) at the Web Conference (27th WWW)*. 2018
- Tobias Käfer, Alexandra Wins and Maribel Acosta. "Modelling and Analysing Dynamic Linked Data using RDF and SPARQL". in: *Proceedings of the 4th International Workshop on Dataset PROFILing and fEderated Search for Web Data (PROFILES) at the 16th International Semantic Web Conference (ISWC)*. 2017. Best Paper
- Andreas Harth and Tobias Käfer. "Towards Specification and Execution of Linked Systems". In: *Proceedings of the 28th Workshop Grundlagen von Datenbanken (GvD)*. GI. 2016, pp. 62–67
- Tobias Käfer, Jürgen Umbrich, Aidan Hogan and Axel Polleres. "Towards a Dynamic Linked Data Observatory". In: *Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW)*. 2012

Acknowledgements

This dissertation would not have been written if it were not for a few people:

First of all, I want to thank Rudi, who built the research group with an amazing spirit in which I conducted most of the research that led to this thesis, York, who took over the group and the role of my doctoral advisor maintaining this spirit, and Andreas, who supervised me and found the right balance between constantly challenging my work and being good fun.

I want to thank Aidan and Jürgen for introducing me to research and for collaborating throughout the years. Of course, this happened under the auspices of and together with Axel, to whom I am on top of that grateful for serving on my committee.

I want to thank my great colleagues in the research group, especially everybody who regularly had cake with me, notably Basil for adding levels of abstraction and puns to everything that deserves it, Maribel for a good laugh about everything that deserves it, and Steffen S. for his advice and teamwork.

It was a pleasure to work with you and thanks for all the fun we had!

The whole process around a dissertation could not be completed without an examination, where I want to thank Alexander Mädche and Martin Ruckes for serving on the committee.

Last but not least I want to thank my friends and family, in particular Sabrina and my parents Ursula and Helmut, who have sustained me throughout the years.

Contents

1. Introduction

The web is the largest information system built by mankind. Since the invention of the web in 1989 [15], the web has grown to a size of 4.53 billion pages[1] (or documents) and is being used by 4.16 billion people[2]. The popular[3] application to use the web is the browser (for instance: Mozilla Firefox[4], Google Chrome[5]), a client[6] application using which users communicate via the internet with web servers[7] to download and read usually one document, before they proceed to the next page, or update the document. However, the web also provides the infrastructure for the communication of user-facing applications, e. g. mobile apps, with servers, e. g. in the cloud. Moreover, communication that is not directly user-triggered, such as in API ecosystems, e. g. around Amazon [63], makes use of the web. Thus, the web has reached the goal to be "a shared information space through which people and machines could communicate" [16].

In this thesis, we want to go beyond the communication aspects (representation of data in documents, transfer of documents via the network) for which web technologies are used traditionally and investigate the notion of behaviour in the context of applications (or agents, or components) that communicate using web technologies.

Behaviour is defined by the Oxford Living Dictionary as:

> *The way in which one acts or conducts oneself, especially towards others*
>
> 1. *the way in which an animal or person behaves in response to a particular situation or stimulus*
> 2. *the way in which a machine or natural phenomenon works or functions*
>
> [14]

This definition of behaviour encompasses both the living (1.) and the artificial and abstract (2.). We adopt these definitions and concretise (2.) for an computer science setting by also regarding as behaviour "the way in which an application works", e. g. a specification of an algorithm to perform a computation. As the subject exhibits behaviour,

[1] `http://www.worldwidewebsize.com/`, visited on 2018-06-10.
[2] `http://www.internetworldstats.com/stats.htm`, visited on 2018-06-10.
[3] We estimate that all of the 4.16 billion users of the internet use a browser, based on a 2015 report by `http://tcrn.ch/1S1N9qo` (visited on 2018-06-10) that 800 million people use Google Chrome on Mobile, which had a market share of 16 % towards the end of 2015 according to `https://analytics.wikimedia.org/dashboards/browsers/#all-sites-by-browser` (visited on 2018-06-10)
[4] `http://www.mozilla.org/firefox/`, visited on 2018-06-10.
[5] `http://www.google.com/chrome/`, visited on 2018-06-10.
[6] A user agent, to be precise in REST terminology.
[7] Origin servers, to be precise in REST terminology.

be it extrinsically as response or intrinsically if it works by itself, the subject is dynamic. In our computer science setting, this dynamics manifests itself in dynamic data, e. g. the values modified by an algorithm. As we consider a networked setting, in which typically multiple components participate, we do not limit our considerations to the behaviour of one component. Instead, we consider applications that compose data and functionality from multiple components.

We consider a particular combination of web technologies, Linked Data, the combination of the transfer protocol HTTP and the data model RDF. The Hypertext Transfer Protocol (HTTP) [75] and the Resource Description Framework (RDF) [47] provide global interoperability in the communication between distributed components on the levels of transferring and representing data with implications on how to compose components into applications:

- *To communicate* requires the ability to successfully transfer, i. e. encode, transmit, and decode messages [190], and to interpret the messages' semantics [107, 191]. Web technologies facilitate this communication using standardisation in the relevant fields. The standardisation bodies W3C and IETF provide open standards for communication roles, message (de/en)coding and semantics (HTTP [75, 76] by the IETF, which builds on TCP/IP [177, 178] for transmission), and message content and semantics, i. e. knowledge representation (RDF(S) [47, 109, 30] by the W3C). Research has accompanied the standardisation, and the stack of web technologies led to the discovery of an architectural style for information systems, Representational State Transfer (REST) [73], behind HTTP.

- *To compose* applications from components requires not only interoperability on the level of communication interfaces. Composition also requires shared and interoperable assumptions about architecture [80]: (1) The assumption which party holds the thread of control of the composed application. In REST, the client application holds the thread of control. Consequently, (2) the assumption about the communication topology in REST is that communication is predominantly star-shaped around the client. (3) The assumption about the dependency between the state of components. REST encourages loose coupling and independent evolvability of components.

To investigate behaviour in the context of Linked Data, we cover the following topics:

- Specification of behaviour in static Linked Data – Our investigations address how to describe "the way in which an application works" using RDF available for access using HTTP.

- Monitoring of behaviour manifested in dynamic Linked Data – We record and analyse the dynamics of Linked Data with and without specifications of how the behaviour should be.

- Execution of behaviour on writeable Linked Data – We investigate how to execute specifications of behaviour that modify RDF data via HTTP.

Hence, we investigate the specification, execution, and monitoring of behaviour in an environment that should readily allow for the composition of applications, as the environment addresses the major interoperability challenges identified in [190, 107, 191, 80]. As we specify behaviour to compose distributed components into applications, we work on the vision of programming-in-the-large [50].

1.1. Background

We next give an overview on the technologies on which Linked Data is based. We present typical applications in the context of the technologies and sketch how they are affected by dynamic data. Moreover, we relate this thesis to the body of knowledge of the corresponding technologies. An in-depth discussion of related work will be given in the individual chapters.

1.1.1. REST, HTTP, and the Web Architecture

REST, short for Representational State Transfer [73], is the architectural style of the web, specifically of Uniform Resource Identifiers (URIs) [21] and HTTP. REST fosters loose coupling between the communicating parties, called web agents [126], by separation of concerns [51] into user agent and origin server. REST encourages stateless communication, which means different communication acts do not depend on each other, as they contain all information required for their understanding in a self-contained manner. In this stateless communication, user agents and origin servers transfer representations (i. e. documents) of state information of resources (i. e. subjects of discourse).

As a communication protocol, HTTP reduces the entropy[8] [203] when it comes to exchanging data by defining a constrained set of operations and corresponding message semantics. The operations in HTTP roughly resemble the basic operations of persistent storage, CRUD (create, retrieve, update, delete) [152]. In other architectural styles however, the understanding of a communication act is complicated by e. g. unclear semantics of a message (update or invocation) in publish/subscribe [62].

REST is flexible when it comes to the format of the representation. Again, standardised representation formats reduce entropy [203], and in REST the format of use can be negotiated between client and server. Different representation formats on the entropy continuum balance expressivity for ease of interpretation[9]. Typical examples include plain text, HTML, JSON, XML, or RDF. Linked Data builds on the data model RDF, where we can add links to machine-processable and formal schema definitions that help to further combat entropy [83, 197]. REST-based interaction with URI-identified resources,

[8] Shannon defined *entropy* as the information content of a message chosen from a set of possible messages. Hence, the entropy is a measure of the uncertainty of the outcome of a choice, or in other words, a measure of unpredictability. With p_i as the probability of an outcome i, the entropy H is defined as $H = -\sum p_i \log p_i$ [190].

[9] `https://www.slideshare.net/rnewton/autonomous-agents-on-the-web-22078931`, visited on 2019/07/23.

during which messages in interoperable data formats are exchanged, is the gist of the Web Architecture [126].

Applications

Traditional applications for REST, such as the *browser* or *mobile apps*, communicate mainly with few resources. *Search engines*, on the other hand, download and index massive amounts of representations of resources and provide the functionality to search the representations [31]. Recent developments use REST when integrating multiple components into one single client application: *Mashups* display different resources in an integrated fashion in a browser [224]. *Workflow management systems* interact with REST resources to obtain data to consider this data in the execution of a workflow [169]. Increasingly, small-scale embedded devices on the Internet of Things provide REST-based access to sensors and actuators [23, 139, 220, 199, 223, 222], where the data from one sensor typically has to be put in context with information from other components. Where applications retain copies of the resource descriptions over an extended time span, the dynamics of the resources can make the data outdated.

Relation to the Thesis

In this thesis, we consider REST-based communication of user agents with multiple resources described in RDF. We apply findings from search engines about the dynamics of the web when investigating the dynamics of Linked Data. When considering workflows to describe behaviour in the context of REST, previous works used the assumptions of traditional workflow management systems to include REST resources. In contrast, we use the assumptions of REST when building a workflow management system.

1.1.2. RDF, Ontologies, and The Semantic Web

The Semantic Web is the vision of an environment of technologies in which data published on the web can be understood by machines [114]. RDF is the data model of the Semantic Web. RDF is graph-based and defines a merge operation for data from different sources and is hence suitable for integration. On the Semantic Web, we want to go beyond this syntactic integration and consider the semantics of the terms. The semantics of terms can be make explicit by describing the terms in ontologies, i. e. formal specifications of how concepts, i. e. classes of terms are interrelated [201]. To specify the semantics of terms, ontology languages such as RDF Schema (RDFS) [30] and the Web Ontology Language (OWL) [113] have been standardised. Being formal, we can leverage ontologies and corresponding reasoning when programming machines to process data while taking the data's semantics into account.

Applications

Consequently, in the seminal article on the semantic web by Berners-Lee et al. [22], ontologies are the key enabler for machine *agents* that act on the behalf of humans.

While such agents have not been realised yet [189, 110], other technologies from the semantic web found widespread adoption: The sheer size of RDF data available has called for applications called *Triple Stores*, i. e. databases that allow for efficient storage and querying of RDF data. Early systems that stood the test of time include Jena [153], Sesame [32], and Redland [12]. The interface to Triple Stores and the query language has been standardised in the SPARQL Protocol and RDF Query Language (SPARQL) [195]. The need for expressing complex relations between terms has led to the development of ontology languages of different expressivity and complexity [30, 85, 157] rooted in logics. Applications that perform corresponding logical inferencing are called *reasoners*. While the seminal article has covered the need for ontologies in-depth, the article is fairly silent about how agents on the web access and change data, and only talks about "meaningful manipulation" [22]. In contrast, Linked Data is all about making semantic data accessible on the web, such that user agents can consume and modify such data.

Relation to the Thesis

In this thesis, we investigate the notion of behaviour in Linked Data. Hence, our results affect systems that work on replicated Linked Data or act in federated settings. Such systems typically include include Triple Stores and reasoners. When analysing dynamics, we operate on RDF without doing reasoning, while we do make use of light-weight reasoning when considering workflows, and our formalisations of behaviour in Linked Data leave room for reasoning. We provide an analogy of a part of our work to Semantic Web agents.

1.1.3. (Read-Write) Linked Data

The Linked Data principles are a canonical way of publishing data on the web [17] that combines the technical web standards for identifying resources (URI), for describing resources (RDF), and for accessing resource representations (HTTP) with the social imperative to provide links to other related data. Following those principles, the Linked Data initiative has published a sizeable amount data on the web: The Linking Open Data cloud diagram[10] lists 1 184 datasets as of 2018-04-30. When also considering the state-changing methods of HTTP, we can manipulate writeable sources on the web [18] and use this Read-Write Linked Data for application integration [120]. Clarifications around the use of HTTP and RDF have been standardised in the the Linked Data Platform (LDP) [196].

Applications

Specialised applications for using Linked Data include *crawlers* [102, 124] for downloading Linked Data, e. g. to put the data in a Triple Store for further processing, and browsers [19] to display and browse Linked Data. Moreover, there are applications that combine downloading, link following, querying and reasoning such as cwm[11], Linked Data-Fu [198],

[10] `http://lod-cloud.net/`, visited on 2018-06-10.
[11] `http://www.w3.org/2000/10/swap/doc/cwm.html`, visited on 2018-06-10.

NautiLOD [77], LDQL [106], LTBQE [104, 103], and Ripple [192]. Such applications so far only consider reading access to Linked Data. Besides generic HTTP clients that send RDF, specialised applications that can write to Linked Data include Linked Data-Fu [198].

Relation to the Thesis

In this thesis, we investigate the notion of behaviour in Linked Data. Consequently, all applications that download Linked Data are affected by the dynamics of Linked Data. We study formal approaches to describe the dynamics of Linked Data. Subsequently, we investigate the dynamics of Linked Data on the Web. Next, we provide a model of computation for Linked Data such that we can specify the behaviour of applications operating on Linked Data. Last, we devise a workflow-based language to specify behaviour for monitoring and execution on Linked Data. The thesis has been carried out in the context of Linked Data-Fu[12], a language and an interpreter for downloading of, querying over, reasoning on, and enacting change to Linked Data, to which this thesis provides a formally grounded operational semantics. Moreover, we use the operational semantics in the execution of behaviour on Linked Data that is given in workflows.

1.2. Research Questions and Contributions

Dynamics and behaviour affect applications of HTTP and RDF individually and in the combination, i. e. Linked Data. Hence we ask the main research questions of this thesis:

Main research question: How can we capture the dynamics of Linked Data to inform the design of applications?

1.2.1. Challenges

Semantic Web research is an interdisciplinary field of research to which many disciplines have contributed. The disciplines range from Hypermedia, Software Engineering, Databases, Distributed Systems, Knowledge Representation to Philosophy. Those contributions led to standards and practices, which have been charted in the so-called "Semantic Web Layer Cake", which the the World Wide Web Consortium (W3C)'s Semantic Web Activity maintained until 2013, see Figure 1.1. This Semantic Web Layer Cake does not contain a notion of behaviour. In fact, HTTP, which we showed introduces dynamics, is not part of the figure. Consequently, the distinction between the communication roles of user agents and servers, which addresses the principle of separation of concerns in HTTP, has largely been overlooked in Semantic Web research. While third-party updates to the Semantic Web Layer Cake have introduced HTTP as a means for data access, e. g. see Figure 1.2 from 2009 and a blog post from 2017 [123], the dynamics introduced by HTTP are still not in the scope of the figures. Hence, the challenge is to find analyses and formalisms that not only have the explanatory power and the expressivity to be relevant in practice, but also respect the established standards and practices and their conceptual foundations.

[12] `http://linked-data-fu.github.io/`, visited on 2018-06-10.

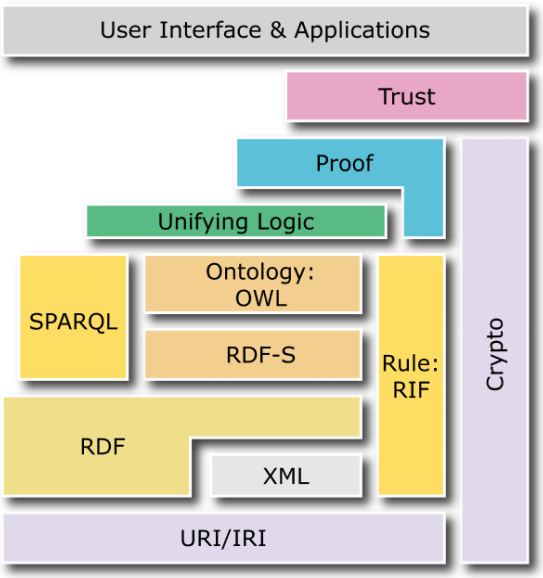


Figure 1.1.: The Semantic Web Layer Cake.

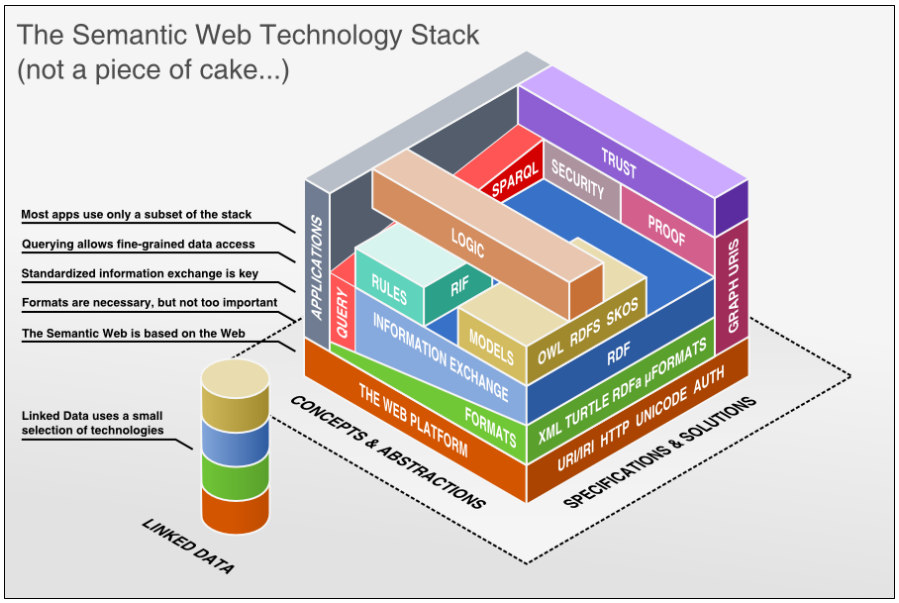


Figure 1.2.: Another layer cake for the Semantic Web [162].

To find suitable analyses and formalisms of behaviour in Linked Data is challenging in itself, as the area of dynamic systems is highly fragmented [145, 127]. Approaches to describe dynamics range from logics, physics, concurrent systems, workflow management, just to name a few. Hence, we need approaches that can be mapped conceptually to the conceptual foundations of Semantic Web Technologies.

1.2.2. Hypotheses

To investigate our main research question, we consider the following hypotheses, which we test in the course of this thesis.

H1. Linked Data is dynamic across application domains.

H2. We can describe the dynamics of Linked Data using formal methods.

H3. We can derive coherent snapshots of Linked Data.

H4. Web technologies can serve as framework for computation and application specification.

1.2.3. Research Questions

In the context of our main research question, we test the hypotheses by answering the following research questions:

RQ1. How can we describe the dynamics of Linked Data using formal methods? → H2

RQ2. How can we construct a corpus to verify and study the dynamics of Linked Data? → H1, H3

RQ3. How can we specify computation using Read-Write Linked Data and rules? → H4

RQ4. How can we combine control flow specifications in workflows with semantic reasoning, and RESTful access? → H4

1.2.4. Contributions

To answer our research questions, we provide our following main contributions. For each contribution, we show the research question the contribution aims to answer and the thus addressed hypotheses.

C1. The Linked Data Transition System → RQ1 → H2

The Linked Data Transition System is a formal abstraction on Linked Data based on Transition Systems to describe the behaviour of Linked Data. This snapshot-based formal abstraction serves as basis for the considerations of the following contributions. We verify the abstraction using the Dynamic Linked Data Observatory.

C2. The Dynamic Linked Data Observatory → RQ2 → H1, H3

The Dynamic Linked Data Observatory is a corpus of Linked Data over time, which consists of more than 300 weekly snapshots of about 100 k Linked Data resources on the web. Using the Dynamic Linked Data Observatory, we shed light on the dynamics of Linked Data on the web by monitoring Linked Data, in particular we investigate the temporal coherence of snapshots.

C3. Statistical analyses for temporally coherent snapshots → H3

We apply statistical methods to investigate the validity of the snapshot-based view on Linked Data using the Dynamic Linked Data Observatory. We found the error that is introduced by networking and processing latencies when compiling the snapshots of the Dynamic Linked Data Observatory, to be small.

C4. ASM4LD → RQ3 → H4

To specify and execute behaviour on Linked Data, we develop Abstract State Machines for Linked Data (ASM4LD), a model of computation for Linked Data based on Abstract State Machines. We found ASM4LD to be Turing-complete and the performance of a corresponding interpreter to be sufficient for application in practice.

C5. WiLD → RQ3 → H4

Based on ASM4LD, we develop Workflows in Linked Data (WiLD), a workflow-based approach to specify, monitor, and execute behaviour on Linked Data. WiLD covers the basic workflow patterns [211] and has been successfully applied with partners from industry.

C6. RWLD-Brick-Benchmark → H4

To evaluate ASM4LD and WiLD, we built a synthetic benchmark for Read-Write Linked Data user agents. The benchmark is set in a building automation setting and simulates parts of the building management systems of one or multiple editions of an office building with about 280 rooms.

We illustrate the layering of the descriptive approaches developed in this thesis (contributions C1, C4, C5) using Table 1.1. The other contributions serve as evaluation.

Evaluations

We evaluate our main contributions as follows:

- The snapshot-based view of the Linked Data Transition System (C1) is evaluated using the statistical analysis for coherence (C3) on the Dynamic Linked Data Observatory (C2). With the snapshot-based view on Linked Data as basis for ASM4LD (C4) and WiLD (C5), we thus also indirectly evaluate the assumptions behind C4 and C5.

Table 1.1.: The layer cake of the approaches in this dissertation to describe dynamics in Linked Data. We build on Read-Write Linked Data, which in turn builds on HTTP. We align our layer cake to the Semantic Web Layer Cake.

Layer	Technology	SW Layer Cake
Workflow Meta Model	WiLD (C5)	RDF-S‡
Model of Computation	ASM4LD (C4)	
Dynamics Model	LDTS (C1)	
Data Model and Access	*[Read-Write Linked Data]*	RDF
Data Access	*[HTTP]*	URI

‡We use a bit more than RDFS reasoning in WiLD, but not full OWL. Specifically, WiLD builds on OWL LD [85].

- The Dynamic Linked Data Observatory is evaluated by pointing to the impact of the data collection and analysis.
- ASM4LD is evaluated formally and using a performance benchmark (C6).
- WiLD is evaluated formally, using a performance benchmark (C6), and by presenting a showcase developed with partners from academia and industry. Additionally, by evaluating WiLD, which is based on ASM4LD and LDTS, we evaluate the whole stack of the descriptive approaches presented in thesis.

1.3. Scope

Our investigations of behaviour covers both the server behaviour that influences client applications (specifying, monitoring) and the behaviour of client applications that influences servers (specifying, executing). In both cases, we consider client applications whose data is composed by communicating with multiple components with Linked Data interfaces.

This thesis operates mostly on HTTP and RDF. Hence, we do not consider behaviour in the levels in the Semantic Web Layer Cake higher than RDF. In fact, higher levels have been addressed by other people using results from this thesis. Yet, we use OWL LD reasoning as means when investigating behaviour. Next to RDF in the layer cake, there are rules. If we consider rules in this thesis, we consider fixed rule sets, i. e. the rule sets are not subject to behaviour. Farther right are cryptographic methods, which we consider out of scope for this thesis. Yet, our findings do support access control and cryptography if it can be transparently layered on top of HTTP, e. g. using HTTPS [182].

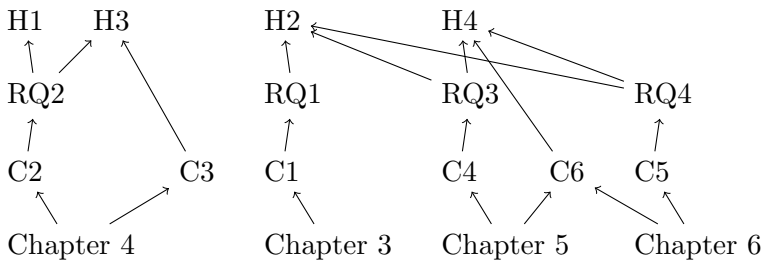


Figure 1.3.: The distribution of the investigation of hypotheses, answers to the research questions, and contributions to chapters in this thesis.

1.4. Structure of the Thesis

The remainder of the thesis is structured as follows. First, in Chapter 2, we give an in-depth treatment of the basic technologies and definitions that are required in the remaining chapters. Then, we present our contributions in the order how the contributions build on each other, see Figure 1.3. As we go, we provide answers to the research questions, and test hypotheses. Last, we conclude in Chapter 7.

1.5. Example

Throughout the thesis, we use the following example: Imagine two devices from the Internet of Things with Linked Data interfaces. Both devices be located in a room. One device has a temperature sensor, such that we can retrieve the current temperature described in RDF using HTTP requests. The second device serves as a switch of two lamps attached, whose state is described in RDF, which we can read and write by sending corresponding HTTP requests. We use the SSN/SOSA ontologies [92] to describe the devices, and the SAREF ontology [49] to describe the sensors, actuators, and the actuators' state. Moreover, we use the FOAF ontology[13] to define the relation between a resource and the document about a resource. We give details on the used URIs and the data obtained by sending an HTTP request to the URIs. We used a set-up that included such devices based on Tessel 2 boards for a demo [133]. Corresponding source code to run on Tessel 2 boards can be found online[14]. We show such devices in Figure 1.4.

Using this example, we describe a small instantiation of the (Read-Write) Linked Data setting. The example is in a networked setting and contains two servers that serve representations of the state of resources in RDF over HTTP. These resources include the documents about the devices and the sensors. The state of two particular resources, namely the documents that describe the two relays for the lamps, is writeable. In RDF, the granularity using which the descriptions of resources are distributed to documents is not specified [221]. We applied a fairly fine-granular distribution such that we do not

[13] `http://xmlns.com/foaf/spec/`, visited on 2018-06-10.
[14] `https://github.com/kaefer3000/t2-rest-relay-climate`, visited on 2018-06-10.

Table 1.2.: URIs of resouces on the two Internet of Things devices for the example, and the content of message bodies in responses to HTTP-GET requests.

URI	Response Message Body Content

`http://t2-relay.example/`	The description of the root resource with a pointer to the overview of the device's modules:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <modules/#it>.`

`http://t2-relay.example/modules/`	The overview over the modules with pointers to the device's modules:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <relay/#it>.`

`http://t2-relay.example/modules/relay/`	The description of the device's module, which points to sub-resources:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <1#it>, <2#it> .`

`http://t2-relay.example/modules/relay/1`	The description of the actuator (analogously for the second switch):
	`<> foaf:primaryTopic <#it> .`
	`<#it> a saref:LightSwitch ;`
	`saref:hasState saref:On .`

(a) On the Internet of Things device with the light switch.

URI	Response Message Body Content

`http://t2-climate.example/`	The description of the root resource with a pointer to the overview of the device's modules:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <modules/#it> .`

`http://t2-climate.example/modules/`	The overview over the modules with pointers to the device's modules:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <climate/#it> .`

`http://t2-climate.example/modules/climate/`	The description of the device's module, which points to sub-resources:
	`<> foaf:primaryTopic <#it> .`
	`<#it> sosa:hosts <temperature#it> .`

`http://t2-climate.example/modules/climate/temperature`	The description of the sensors:
	`<> foaf:primaryTopic <#it> .`
	`<#it> a saref:TemperatureSensor ;`
	`rdf:value 20 .`

(b) On the Internet of Things device with the temperature sensor.

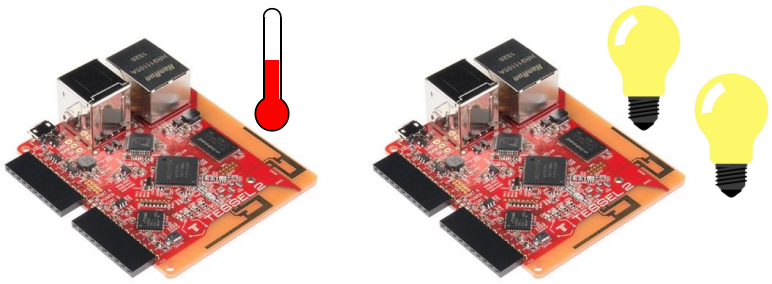


Figure 1.4.: Two Internet of Things devices, which we use in our examples. One has a thermometer attached, the other controls two lamps via relay switches. Images from SparkFun, Pixabay, and Wikimedia Commons.

overwrite the state of many resources using one request. Using the example, we can illustrate how Linked Data changes, how to derive Linked Data snapshots from different resources, how to follow links, and how to change Linked Data. The example does not show how to follow redirects. We give more details on the example as we introduce of technologies using the example. The generalisation to larger scenarios is straight-forward.

2. Preliminaries

In this chapter, we lay out foundational definitions that most of this thesis' chapters require: (Read-Write) Linked Data, URIs, HTTP, RDF Graphs and datasets. We give definitions that are relevant only in certain chapters in the corresponding places: model-theoretic semantics for RDF (Section 5.6.2), Notation3 and rules (Section 5.4.1) as well as terminology around dynamics and workflows (Section 6.5).

2.1. (Read-Write) Linked Data

The Linked Data principles are a set of practices for data publishing on the web [17]. They encourage the use of web standards for identifying and accessing data and to link the data with data from other providers. The Linked Data principles are:

> 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 (RDF*, SPARQL)*
> 4. *Include links to other URIs. so that they can discover more things.*
>
> [17]

While the Linked Data principles are concerned with accessing data, Read-Write Linked Data generalises the principles and also encourages writing access [18].

2.2. Resources and URIs

The following definitions build on the notions of resources and URIs, which are identifiers for resources.

Definition 1 (URI, Resource, Scheme, Scheme-specific part)**.** *Uniform Resource Identifiers (URIs) are character sequences that serve as names on the web. A resource identified by a URI may be anything, abstract or physical. URIs consist of a scheme and a scheme-specific part, divided by a colon [21].*

The scheme determines how the scheme-specific part is to be interpreted. In this thesis, we assume URIs with the HTTP schemes (http and https), which indicate that HTTP

interaction may be possible with the URI. URIs are absolute or relative. If a scheme allows for expressing a hierarchical structure such as the HTTP schemes, then relative URIs allow for giving URIs that are relative to other URIs in the hierarchical structure.

Example 1 (URIs). *We identify the Internet of Things devices using URIs*[1]*:*

- `http://t2-climate.example/#it`
- `http://t2-relay.example/#it`

We also identify the modules on the Internet of Things devices using URIs:

- `http://t2-climate.example/modules/climate/#it`
- `http://t2-relay.example/modules/relay/#it`

and the sensors and actuators on the modules:

- `http://t2-climate.example/modules/climate/temperature#it`
- `http://t2-relay.example/modules/relay/1#it`

Definition 2 ((Non) Information Resource, Collection Resource). *An Information Resource is a resource that provides a non-empty message body in HTTP responses with status code 200 to HTTP-GET requests (cf. Section 2.3). Non-Information Resources are all other resources. Collection resources are collections of resources.*

Example 2 (Collection Resource). *The set of modules on the Internet of Things devices are collection resources:*

- `http://t2-climate.example/modules/`
- `http://t2-relay.example/modules/`

Similarly, the set of relays on the relay module is a collection. Using the relay, we to turn a light on and off: `http://t2-relay.example/modules/relay/`

Example 3 ((Non) Information Resource). *Although the URI* `http://t2-climate.example/modules/climate/temperature#it` *identifies the temperature sensor, we cannot do an HTTP request to said URI. Hence, this URI is not an information resource. However, we can make a request to* `http://t2-climate.example/modules/climate/temperature`*, which is the URI of the document to describe the temperature sensor. Said URI is indeed an information resource, if we assume that the server implementation on the Internet of Things device answers HTTP-GET requests to said URI with status code 200 and a non-empty message body.*

[1] We use the top-level domain `.example` in our examples as put forward in [59].

Table 2.1.: HTTP status code classes and explanations [76].

Range	Class	Explanation
1xx	Informational	The request was received, processing continues
2xx	Successful	The request was received and can be successfully answered
3xx	Redirection	The request needs further client action for completion
4xx	Client Error	The request cannot be fulfilled due to a client error
5xx	Server Error	The request cannot be fulfilled due to a server error

2.3. Hypertext Transfer Protocol (HTTP)

URIs with one of the HTTP schemes indicate that communication with the resource may be possible [75]. In the Hypertext Transfer Protocol [75] (HTTP), communication happens in request/response pairs. We call the party initiating a request *user agent*, and the responding party *origin server*.

Definition 3 (HTTP Request, Response, Method, Target, Status Code)**.** *A HTTP message is a tuple $\langle S, H, B \rangle$, where S is the mandatory start line, H is an optional list of header name/value pairs, and B is the message body, also optional. A HTTP request is a HTTP message in which the start line S consists of a request line (with the HTTP method, the target of the request, and the version information). In a HTTP response message, the start line S consists of a numerical HTTP status code along with a textual explanation, and information on HTTP version used.*

An HTTP header contains meta information on the HTTP message. For instance, the resource representation format can be negotiated in a process called *content negotiation* [76] between client and server using the `Accept` header in the request, which lists the preferences of the client, and the `Content-Type` header in the response, which states the representation format the sever chose. We present the `Location` header in the following when discussing HTTP status codes.

Status codes are three-digit integers, where the first digit determines the status code class. The HTTP specification distinguishes the classes in Table 2.1. The responses with 3xx status code typically contain a `Location` header with a URI to which the server redirects the client for the next request.

The HTTP method in an HTTP request states the purpose of the request for the user agent, and determines the semantics of the message bodies in the corresponding request/response pair. Of the HTTP methods in the standard, we consider the most prominent methods: GET, PUT, POST, DELETE. Those methods can be roughly mapped to CRUD (create, retrieve, update, delete), the basic operations of persistent storage [152]. We next present the HTTP methods that we consider in this thesis:

- A user agent uses a GET request to ask the server for a representation of the current state of the target resource.

Table 2.2.: The semantics of HTTP message bodies [76, 196] ...

(a) ... in requests.

Method	The Message Body Contains ...
GET	Nothing
PUT	State of the resource
POST	Arbitrary data or state of a resource
DELETE	Nothing

(b) ... in responses with certain status codes.

Method	Status Code Range	The Message Body Contains ...
GET	2xx	State of the resource
PUT	2xx	State of the resource or nothing
POST	2xx	State of the request or nothing
DELETE	2xx	State of the request or nothing
any	Non-2xx	State of the request or nothing

- A user agent uses a PUT request to tell the server to overwrite the current state of the target resource using the representation supplied.
- A user agent uses a DELETE request to order the server to delete the target resource.
- The use of a POST request depends on the type of the target resource. The standard discusses a few types of functions including:
  - If the target resource represents a collection, a user agent uses the POST request to add a resource to the collection. The supplied representation describes the resource to be created. We call this function POST-append in this thesis. In line with level 2 on the Richardson Maturity Model [78], we only consider in this function of the POST request in this thesis.
  - If the target resource represents a data-handling process, a user agent can supply data to the process and gets an answer in return. We call this function POST-RPC in this thesis.

The semantics of the message bodies of messages in requests and responses under the different HTTP methods and status codes is summarised in Table 2.2. As we consider Linked Data in this thesis, we assume the message bodies to be in RDF. Note that the Linked Data Platform specification [196] provides clarifications regarding HTTP when RDF data is transferred. For instance, the response to a successful POST-append request should not be expected to contain a message body.

HTTP methods have different properties. For instance, the property a method of being *safe* means that a client sending a request with a safe method does not expect change

Table 2.3.: Properties of HTTP methods.

Method	Safe?	Idempotent?
GET	X	X
PUT		X
POST		
DELETE		X

on the origin server caused by the request. A request with an *idempotent* method sent multiple times *ceteris paribus* shall have the same result as sending the request once. We summarise the properties of different request methods in Table 2.3.

2.4. Resource Description Framework (RDF)

As indicated by the Linked Data principles, we assume that the representation of the state of resources is given in an RDF Graph.

Definition 4 (RDF Term, Triple, Graph)**.** *The set of RDF terms consists of the set of URIs U, the set of blank nodes B and the set of RDF literals L, all being pairwise disjoint. A tuple $\langle s, p, o\rangle \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple, where s is the subject, p is the predicate, and o is the object of the triple. A set of triples is called RDF Graph.*

In this thesis, we use the Turtle syntax[2] to describe RDF Graphs [180].

Example 4. *We can interact with the Internet of Things device using HTTP. For instance, we can send an HTTP-GET request to the root resource. We retrieve a successful response with a message body in Turtle format, see Figure 2.1.*

To be able to talk about the state representations retrieved from multiple resources, i. e. multiple RDF Graphs, we introduce the notion of an RDF Dataset [47].

Definition 5 (Named Graph, RDF Dataset)**.** *Let G be the set of RDF Graphs and let U be the set of URIs. A pair $\langle g, u\rangle \in G \times U$ is called a named graph. An RDF Dataset consists of a (possibly empty) set of named graphs (with distinct names) and a default graph $g \in G$ without a name.*

While the standard only discusses said structure of an RDF Dataset, the semantics of RDF Datasets have not been standardised. Zimmermann discusses different semantics for an RDF Dataset [225]. In this thesis, we adopt the semantics discussed as "Named graph[s] are in a particular relationship with what the graph name dereferences to" in Section 3.5 of [225]: At one point in time, all names in the graph have been dereferenced and the retrieved RDF Graphs have been put into the RDF Dataset at the corresponding

[2] We use prefix definitions as provided by `http://prefix.cc/`, visited on 2018-06-10.

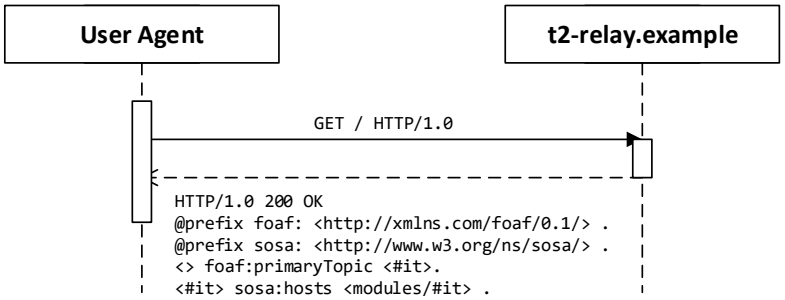


Figure 2.1.: UML Sequence Diagram of an HTTP-GET request from a user agent to the server at `t2-relay.example`, which runs one of the Internet of Things devices in our example. The server device answers with a successful response with the message body in Turtle.

graph name. To talk about the state of resources at different system states, we introduce an index t, which denotes the system state to which an RDF Dataset D refers, thus yielding D_t. We assume discrete time represented as monotonically increasing integers.

Example 5 (RDF Datasets)**.** *We show RDF Datasets for two points in time for three resources from the Internet of Things device with the light, see Figure 2.2. Note that the graph at the bottom changes.*

Figure 2.2.: RDF Datasets for two points in time of three resources from the Internet of Things device with the light.

Name	Graph
`http://t2-relay.example/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <modules/#it> .`
`http://t2-relay.example/modules/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <relay/#it> .`
`http://t2-relay.example/modules/relay/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <1#it> , <2#it> .`

`http://t2-relay.example/modules/relay/1`	`<> foaf:primaryTopic <#it> .`
	`<#it> a saref:LightSwitch ;`
	`saref:hasState saref:Off .`

(a) D_0

Name	Graph
`http://t2-relay.example/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <modules/#it> .`
`http://t2-relay.example/modules/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <relay/#it> .`
`http://t2-relay.example/modules/relay/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <1#it> , <2#it> .`

`http://t2-relay.example/modules/relay/1`	`<> foaf:primaryTopic <#it> .`
	`<#it> a saref:LightSwitch ;`
	`saref:hasState saref:On .`

(b) D_1

3. A Formal Basis for Dynamic Linked Data

Parts of this chapter have been published in:
▷ Andreas Harth and Tobias Käfer. "Towards Specification and Execution of Linked Systems". In: *Proceedings of the 28th Workshop Grundlagen von Datenbanken (GvD)*. GI. 2016, pp. 62–67.

In this chapter, we propose the Linked Data Transition System, a formal abstraction on Linked Data over time. To look at Linked Data over time makes intuitively sense at least for the following two reasons: (1) As HTTP URIs can represent resources that stand for things in the real world, the representations retrieved at different points in time can be assumed to vary. (2) Moreover, we can change Read-Write Linked Data using state-changing HTTP requests. The Linked Data Transition System describes Linked Data as snapshots at different points of time. This snapshot-based view on Linked Data is the basis for the considerations in the subsequent chapters: In Chapter 4, we analyse changes between different Linked Data snapshots and investigate the temporal coherence of snapshots to evaluate the notion of Linked Data snapshots. In Chapter 5, we interpret Linked Data snapshots as states of algorithm runs. In Chapter 6, we use workflows that operate on Linked Data snapshots and whose workflow instance state is maintained in Linked Data snapshots.

3.1. The Linked Data Transition System

In the Linked Data Transition System, we take an omniscient view on states, requests and responses in the system, and describe the state of relevant Linked Data sources at one point in time as RDF Dataset. The graph names in the RDF Datasets in the Linked Data Transition Systems be URIs of information resources. The RDF Graphs in the RDF Datasets are then the graphs obtained by dereferencing the URI at the time of the RDF Dataset, cf. Section 3.5 in [225]. As an omniscient observer, we know when a change occurs in the system. Then, we compile a new RDF Dataset. While we here take an omniscient view and thus present an idealised view, in practice, a user agent has to make HTTP-GET requests to create the RDF Datasets (e. g. in Chapters 4, 5, 6). Compared to the idealised view, the user agent perspective introduces two inaccuracies:

- Network latencies can make the RDF Dataset temporally incoherent, i. e. the one point in time is in reality different points in time for different URIs
- When operating in a periodic fashion, the sampling rate may be too low such that the user agent misses system states. On the other hand, if the sampling rate is higher than the rate of change, we have RDF Datasets in the Linked Data Transition System with no change in between.

The changes may be caused by one or many HTTP requests (e. g. in the case of Read-Write Linked Data), or by the environment (e. g. if the real-world things change). Thus, the transitions in the Linked Data Transition System are sets of requests. We hence define the Linked Data Transition System as follows:

Definition 6 (Linked Data Transition System)**.** *Let $Req, Resp$ be the set of all HTTP request/response pairs. A Linked Data Transition System is a pair $(S, \rightarrow)$:*

- *A set of RDF Datasets S representing resource states.*
- *The transition relation $\rightarrow$ over $S \times 2^{Req,Resp} \times S$. We can contrast the current state s_t with the next state s_{t+1}, given a transition occurs. A transition consists of a set of request/response pairs.*

The states S are the states of all resources in the system. We write $s_0, s_1 \ldots s_n$ to denote datasets at different time points. With $\rightarrow$ denoting the transition relation, we write $(s_o, r_o, s_1) \in \rightarrow$ as $s_o \xrightarrow{r_0} s_1$. We can define the history of a system by a sequence $s_o \xrightarrow{r_0} s_1 \xrightarrow{r_1} s_2 \ldots$, where s_o, s_1, s_2 are states, and r_0, r_1 are sets of request/response pairs.

Based on the Linked Data Transition System, we can now formally illustrate the semantics of different HTTP request methods. For instance, a successful HTTP-GET request trivially transitions to the same state. Given robust server implementations, unsuccessful requests also transition to the same state. Less trivial are the semantics of successful unsafe requests (we denote a "don't-care" as "$\cdot$" and the empty set as "$\emptyset$"). Let u be a URI denoting some resource.

- `PUT`: Let r be a request/response pair, and B a message body. $r := (\langle \texttt{PUT}\ u, \cdot, B \rangle, \langle \texttt{204 No Content}, \cdot, \emptyset \rangle)$. Applying r to state S_t yields S_{t+1}, where in S_{t+1} the triples belonging to u are B.
- `POST`: We assume in this thesis the `POST` request to append to a resource, hence we discuss the `POST` request below.
- `DELETE`: Let r be a request/response pair. $(\langle \texttt{DELETE}\ u, \cdot, \emptyset \rangle, \langle \texttt{204 No Content}, \cdot, \emptyset \rangle)$. Applying r to state S_t yields S_{t+1}, where in S_{t+1}, there is no representation available at u.

Now, let uc be a URI from U identifying a collection resource.

- `PUT`: `PUT` is not possible on collection resources (we assume collections are managed by the server). LDP does not require the `PUT` request to be supported in the context of collections either.
- `POST`: Let r be a request/response pair, B be a message body, and H be a set of headers. $r := (\langle \texttt{POST}\ uc, \cdot, B \rangle, \langle \texttt{201 Created}, H, \cdot \rangle)$. Applying r to state S_t yields S_{t+1}, where in S_{t+1} there are additional triples (those in B) related to a newly

created[1] URI *ue* that is linked to *uc*. Also, *uc* is different now, as the link to the newly created resource identified via URI *ue* is part of the state of the collection resource. The set of headers H contains *ue* as the value of the `Location` header. In compliance with LDP, we assume no representation in the body of the response.

- `DELETE`: Let r be a request/response pair ($\langle$`DELETE` $uc, \cdot, \emptyset\rangle$, $\langle$`204 No Content`$, \cdot, \emptyset\rangle$). Applying r to state S_t yields S_{t+1}, where in S_{t+1} there is no representation available for *uc*. Whether deleting the collection resource *uc* also affects its associated element resources depends on the type of the relation between the collection and its elements (cf. the container types specified in LDP).

Example 6 (Linked Data Transition System)**.** *We again show the two RDF Datasets from example 5 of two points in time for four resources from the Internet of Things device with the light switch, see Figure 3.1. This time, the two RDF Datasets are part of a Linked Data Transition System, where the two RDF Datasets differ in the graph at* `http://t2-relay.example/modules/relay/1`*: In* D_0*, the light switch is set to off, whereas in* D_1*, the light switch is set to on. The change in graphs in Figure 3.1 is caused by a successful HTTP-PUT request, which transforms* D_0 *into* D_1*.*

3.2. Related Work

Stadtmüller et al. defined the REST state transition system (RSTS) [198] (called Linked Data State Transition System in [197]), which captures a similar notion as the the LDTS. However, the RSTS takes a different standpoint: The LDTS regards the state of set of Linked Data resources irrespective of their component and describes as a omniscient observer how the combined state of the Linked Data resources evolves, using HTTP requests or not. The RSTS however models one component, the component's internal state, and the component's reactions, similar to an acceptor of HTTP requests that answers using HTTP responses. As a formal difference, the LDTS uses RDF Datasets to describe states, whereas the RSTS uses RDF Graphs.

Similarly, the main difference to Hernández et al.'s formal account of semantic RESTful APIs [111] is that their account is concerned with only one component, and the component's inner workings. For the account, Hernández et al. use a combination of the π-calculus [155, 156] and triple-space computing [65], an extension of tuple-space computing [82] to RDF.

3.3. Conclusion

In this chapter, we presented the Linked Data Transition System (LDTS), a formal account of dynamic Linked Data. As a model, the LDTS is an abstract view on Linked Data over time. The snapshot-based view on Linked Data in LDTS is the foundation for

[1] Or taken from a pre-filled reservoir of URIs, for example as defined in Abstract State Machines.

Figure 3.1.: Two RDF Datasets, D_0 and D_1, as states in a Linked Data Transition System and an HTTP-PUT request/response pair as transition. The request is depicted on the left of the arrow, the response on the right of the arrow. We highlight where the change happens by underline.

Name	Graph
`http://t2-relay.example/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <modules/#it> .`
`http://t2-relay.example/modules/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <relay/#it> .`
`http://t2-relay.example/modules/relay/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <1#it> , <2#it> .`
`http://t2-relay.example/modules/relay/1`	`<> foaf:primaryTopic <#it> .` `<#it> saref:hasState saref:Off .`

(a) D_0

(PUT http://t2-relay.example/modules/relay/1,
∅,
`<> foaf:primaryTopic <#it> .`
`<#it> saref:hasState saref:On .`)

↓

(204 No Content,
∅,
∅)

Name	Graph
`http://t2-relay.example/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <modules/#it> .`
`http://t2-relay.example/modules/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <relay/#it> .`
`http://t2-relay.example/modules/relay/`	`<> foaf:primaryTopic <#it> .` `<#it> sosa:hosts <1#it> , <2#it> .`
`http://t2-relay.example/modules/relay/1`	`<> foaf:primaryTopic <#it> .` `<#it> saref:hasState saref:On .`

(b) D_1

the considerations of the subsequent chapters. For the snapshots, which constitute the states in the LDTS, the model abstracts from the network access required to assemble the snapshots. In reality, the snapshots are subject to synchronisation issues introduced by network latencies. We hence experimentally investigate the validity of the snapshot-based view in the presence of network access in the next chapter, specifically in Section 4.9.4.

4. The Dynamics of Linked Data on the Web

Parts of this chapter have been published in:
▷ Tobias Käfer, Alexandra Wins and Maribel Acosta. "Modelling and Analysing Dynamic Linked Data using RDF and SPARQL". in: *Proceedings of the 4th International Workshop on Dataset PROFILing and fEderated Search for Web Data (PROFILES) at the 16th International Semantic Web Conference (ISWC)*. 2017. Best Paper,
▷ Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne and Aidan Hogan. "Observing Linked Data Dynamics". In: *Proceedings of the 10th European Semantic Web Conference (ESWC)*. 2013, pp. 213–227,
▷ Tobias Käfer, Jürgen Umbrich, Aidan Hogan and Axel Polleres. "Towards a Dynamic Linked Data Observatory". In: *Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW)*. 2012.

In this chapter, we experimentally investigate the dynamics of Linked Data. We verify the hypothesis that Linked Data is dynamic and look at the validity of the snapshot-based view on Linked Data, which serves as basis for the subsequent chapters. For these investigations, we set up a long-term study of Linked Data on the web, the *Dynamic Linked Data Observatory*. We motivate the study by presenting challenges in building applications for Linked Data induced by the dynamics of Linked Data. We describe the design of the study, and present findings on the dynamics of Linked Data on the web. Moreover, we investigate the suitability of Semantic Technologies for conducting our analyses.

Example 7. *(Dynamic Linked Data on the web) In our example for the RDF Dataset, we looked at four resources of an Internet of Things device at two points in time. The RDF Datasets of Figure 2.2 can be imagined as being created by making HTTP-GET requests to the four resources at two points in time and by recording the data from the responses. In the Dynamic Linked Data Observatory, we actually create RDF Datasets by doing the HTTP-GET requests to roughly 10^5 resources from the web. So far, we have created RDF Datasets for around 300 points in time over a time span of six years.*

4.1. The Need for a Dynamic Linked Data Observatory

The dynamics of Linked Data affects both Linked Data consumers and providers as they build applications. We discuss challenges induced by the dynamics of Linked Data, and derive use-cases for our analyses of dynamic Linked Data. The use-cases build on previous investigations by Umbrich et al. [208].

4.1.1. The Need of Linked Data Consumers

Consuming Linked Data involves reasoning on and querying of data downloaded from the web. We identify two main challenges induced by changing data and present corresponding use-cases:

Freshness of Caches and Indices To improve performance, reasoning and querying systems employ caches and indices:

Caches improve performance by saving user agents from potentially expensive network access. To this end, caches replicate remote data locally in a way that is transparent for an user agent's application logic. For instance, HTTP Caches are relevant on the Semantic Web for approaches that use HTTP-GET requests for accessing RDF Graphs. Those approaches include live-querying approaches such as Linked Data-Fu [198], NautiLOD [77], LDQL [106], LTBQE [104, 103], scripting approaches such as Ripple [192], and SPARQL engines that retrieve RDF via HTTP-GET in URIs in FROM clauses such as roqet[1] and Virtuoso[2]. Moreover, other approaches use HTTP-GET requests to transfer intermediary results in query processing, e. g. Triple Pattern Fragments [216] (typically with HDT back-end [67]) and federated SPARQL engines [64, 179].

In contrast to caches, indices are part of a user agent's application logic. Indices improve performance by not just removing the need for network access, but also by locally performing pre-computations. Systems with indices on the Semantic Web where index freshness can be an issue include triple stores that replicate data, e. g. mirrors of Linked Data in private deployment, such as a local copy of DBpedia [8], or public semantic web search engines such as SWSE [117], the LOD Laundromat [13], Sindice [204], Swoogle [53], Watson [48], and the LOD Cache[3]. Moreover, there are hybrid approaches that use indices for some sources, and do live-querying on others [207].

For up-to-date results with both caches and indices, changes in the original sources have to be reflected locally. Hence, knowledge about the dynamics of data would allow for the design of applications with caches and indices that can do, e. g.:

Focussed Synchronisation Applications that build on pre-computed indices can determine for which sources they update their indices at which intervals.

Smart Caching Applications that build on caches can determine which sources to cache and for how long cached data can be assumed to be valid.

Adaptive Processing Hybrid applications that involve both, indices and caches, can choose the appropriate approach per source.

Temporal Coherence of Snapshots Approaches that integrate and reason over data assume the data to stem from one point in time. Yet, in a distributed and

[1] `http://librdf.org/rasqal/roqet.html`, visited on 2018-06-10.
[2] `https://virtuoso.openlinksw.com/`, visited on 2018-06-10.
[3] `http://lod.openlinksw.com/sparql`, visited on 2018-06-10.

networked setting, data is downloaded from different sources, where the download from different sources may exhibit different latencies. In the presence of dynamic sources, those latencies can make a combined dataset temporally incoherent, i. e. the dataset contains data about different points in time from different sources [194].

Affected systems on the Semantic Web include those approaches to benefit from HTTP-GET caches listed above that obtain data from multiple sources. To that list, we add SPARQL federated querying that uses HTTP-POST for querying [64, 179].

Optimised Download Schemes Applications involving data form multiple sources can influence the downloading process, e. g. by ordering the sources to improve temporal coherence [194].

Quantification of Error Applications involving data form multiple sources can quantify the error introduced by network latencies, which can serve as input for probabilistic reasoning.

4.1.2. The Need of Linked Data Publishers

We identified two challenges in Linked Data publishing introduced by the dynamics of own data and of the data of third parties.

Link Maintenance When publishing Linked Data, publishers are encouraged to put links into their data, to where more information can be found [17]. If the ends to those links are on domains outside of the control of the publisher, dead links may occur after some time, or the ends may no longer be appropriate for linking after changes to the data [89, 176]. Furthermore, novel sources may serve as useful targets to link [218]. Information about dynamics could be built into applications that help publishers to decide how frequently their link-sets need updating depending on, e. g. the domain they target or the type of link.

Versioning When changes are made to Linked Data, versioning [140] should be applied to make sure that other parties that rely on the data to be stable are not negatively affected. For instance, vocabularies are Linked Data which is particularly often re-used and thus vocabularies would be obvious candidates for Linked Data where versioning should be applied. Insights into the dynamics of Linked Data and how changes propagate can e. g. help in developing versioning methods for Linked Data publishing applications.

4.2. Research Questions

The use-cases call for investigations on two levels:

- On the **physical level** of communication, our investigations are mainly concerned with the transfer of documents in HTTP messages. We look at changes only at

the granularity of a document, i. e. we only check whether the RDF Graphs in two documents are different, but do not look at the triples that constitute the difference. As we operate on level of documents, findings on the physical level can be easily contrasted with investigations of the web of documents with content of other types, e. g. HTML.

- On the **logical level** of communication, we look into the content of the documents transferred, i. e. at the triples and terms in the RDF Graphs.

We next present the research questions for our analyses on both levels.

On the Physical Level

Change Frequency: Can we model change frequency of documents with mathematical models and thus predict future changes?

Change Patterns: Can we mine patterns that help to categorise change behaviour?

Lifespan: What is the lifespan of Linked Data documents?

Stability: How stable are Linked Data documents in terms of HTTP accessibility?

Domain-dependent Changes: Do we observe a variation or clustering in dynamics across different domains?

Temporal Coherence: Can we achieve temporally coherent snapshots of Linked Data?

On the Logical Level

Degree of Change: If a document changes, how much of its content is updated?

Growth Rate: How fast is the Linked Data on the web evolving?

Structural Changes: Do we observe any changes in the structure of the network formed by links?

Change Triggers: Can we find graph patterns that trigger or propagate changes through the network?

Vocabulary-dependent Changes: Do we observe different change patterns for data using certain vocabularies, classes or properties?

Vocabulary Changes: How do the semantics of vocabulary terms evolve over time?

4.3. Challenges

With the identified use-cases and research questions for analyses not limited to an application architecture, a domain, or to consumers or producers of Linked Data, we are facing the following conflicting desired properties in our study design. We come back to these trade-offs as we present the set-up of the Dynamic Linked Data Observatory.

First of all, with the aim of building a working system that runs on commodity hardware, feasibility becomes a concern. Feasibility is obviously conflicting with the following desired properties of a data collection:

General-purpose suitable to study for a wide range of interested parties

Broad capturing a wide selection of Linked Data domains

Substantial the number of documents monitored should allow for deriving confident statistical measures

Granular & frequent offering detailed data on sources

Given resource constraints, the following points need to be balanced for the monitoring and crawling scheme:

Contiguous allowing for comparison of sources over time

Adaptive able to discover the arrival of new sources

In practice, there also other considerations to take into account such as crawling in a **polite** fashion [144], i. e. not to overload remote servers when collecting data.

Thus, the composition of the corpus is a research problem in itself and hence we address RQ2 first: *How can we construct a corpus to verify and study the dynamics of Linked Data?*

4.4. Contributions

Our first contribution is the design of the Dynamic Linked Data Observatory. We have been running this observatory since May 2012 and have collected a significant corpus of data that captures the inherent dynamics of Linked Data. We make all data available for the community; please see `http://purl.org/dyldo/` for up-to-date weekly snapshots.

Our second contribution is the method to analyse dynamic Linked Data using Semantic Web Technologies. The approach to use declarative specifications of analyses allows for concise specifications of analyses, which ease validation.

Our third contribution is the comprehensive analysis of the dynamics of Linked Data. We shed light on the dynamics of Linked Data on the physical levels of accessing documents and on the logical level of RDF documents with the aim of providing insights that inform the design of Linked Data applications.

4.5. Related Work

Few papers specifically analysed RDF or Linked Data dynamics, especially previous to our work on the Dynamic Linked Data Observatory. We discuss those works in Section 4.5. Our work on the Dynamic Linked Data Observatory sparked interest in the dynamics of Linked Data, and other researchers used our dataset for their investigations. We summarise those works when we discuss the impact of our work in Section 4.11.

In contrast to RDF and Linked Data, the dynamics on the HTML web has been subject to a considerable amount of scholarly investigation. We survey findings of those investigations in Section 4.5, relate the findings to our research questions, and take a closer look at the data collection to ground our set-up in literature.

4.5.1. Studies of the Semantic Web and the Semantic Web's Dynamics

Klein et al. classified changes to ontologies [141], without looking at the data access. Popitsch and Haslhofer [176] propose DSNotify to help maintain links between datasets, but only have knowledge of DBpedia dynamics. Umbrich et al. analysed changes in documents over 24 snapshots of RDF web data [205]; however, the coverage of each snapshot varied and the analysis was rather "best-effort", whereas our aim is to provide and analyse a broad and contiguous dataset. Authors from the same group also showed that two centralised query indexes of Linked Data (OpenLink's LOD Cache[4] and Sindice's SPARQL endpoint[5]) often return stale results [207]. Passant et al. investigated how to evaluate continuous queries over SPARQL endpoints [168]. Recently, Schon and Staab investigated dynamics of DL-knowledge bases behind a SPARQL endpoint [188].

While our aim is to compose and analyse a corpus of Linked Data on the web, other approaches survey and monitor other aspects of the Semantic Web. For example, Auer et al. built LODStats to collect statistics about RDF available on the web in the form of Linked Data, as compressed files, or behind SPARQL endpoints [9]. The authors provide statistics such as the depth of the class hierarchy, or number of triples, literals, and blank nodes. LODStats ran from 2012 until 2016. SPORTAL by Hasnain et al. provides a catalogue of SPARQL queries to compute statistics (including VoID descriptions [2]) about SPARQL endpoints [108]. SPARQLES by Aranda et al. focuses on monitoring publicly available SPARQL endpoints [6]. The authors provides a set of predefined queries to inspect the support of SPARQL features and performance of endpoints and monitor SPARQL endpoints in this fashion since 2013. The datasets for the Billion Triple Challenge (BTC) of different years from 2008[6] to 2014 [94, 95, 96, 97, 129] provide crawls of Linked Data. Yet, the BTC datasets do not provide contiguous monitoring, as every year the crawls' seed list was compiled differently, also the frequency of the monitoring, about yearly, is not sufficient for our investigations.

4.5.2. Studies of the Web and the Web's Dynamics

The study of the evolution of the web (of Documents) and its implicit dynamics reaches back to the proposal of the first generation of autonomous World Wide Web spiders (aka. crawlers) around 1995. Bray [25] published one of the first studies about the characteristics of the web and estimated its size in 1996. Around the same time, web indexes such as AltaVista or Yahoo! began to offer one of the first concrete use-cases for understanding the change frequency of web pages: the efficient maintenance of search engine indexes. In 1998, Coffman et al. [46] proposed a revisiting strategy for web crawlers to improve the "freshness" of an index. This work was continuously improved over the subsequent years with additional experimental and theoretical results provided by Brewington and Cybenko [28, 29], Lim et al. [149], Cho and Garcia-Molina [44, 43], Fetterly et al. [72], Ntoulas et al. [163], and Koehler [143] amongst others.

[4] `http://lod.openlinksw.com/sparql`, visited on 2018-06-10.
[5] `http://sparql.sindice.com/`, visited on 2013-03-12, unavailable as of 2018-06-10.
[6] The data set from 2008 is unavailable as of 2018-07-01.

Based on large data collections, these papers presented theory and/or empirical analyses of the HTML web that relate closely to the dynamicity questions we highlight. For example, various authors discovered that the change behaviour of web pages corresponds closely with – and can be predicted using – a Poisson distribution [28, 29, 44, 43]. Relating to high-level temporal change patterns, Ntoulas et al. [163] analysed the different frequency of updates for individual weekdays and working hours. The same paper also empirically estimated the growth rate of the web to be $\sim 8\,\%$ new content every week, and regarding structural changes, found that the link structure of the web changes faster than the textual content by a factor of 3. Various authors found that with respect to the degree of change, the majority of changes in HTML documents are minor [149, 72, 163]. Loosely related to change triggers, Fetterly et al. [72] found that certain parties simulate content changes to draw the attention of search engines. Regarding domain dependent changes, various authors also showed that change frequencies vary widely across top-level domains [72, 44, 28, 29].

Relating to use-cases for studying the dynamics of web documents, a variety have been introduced through the years, including (i) the improvements of web proxies or caches looked at by, e. g. Douglis et al. [58], (ii) efficient handling of continuous queries over documents [167]. We refer interested readers to the excellent survey by Oita and Senellart [164], which provides a comprehensive overview of existing methodologies to detect web page changes, and also surveys general studies about web dynamics. An earlier survey by Ke et al. [136] addresses the implications of web dynamics to search engines.

4.6. The Set-up of the Dynamic Linked Data Observatory

To investigate the research questions on the dynamics of the Linked Data on the web (Section 4.2), we set up a long-term monitoring study, the Dynamic Linked Data Observatory. In this section, we describe how we set up the Dynamic Linked Data Observatory in 2012. We first derive a definition of what, i. e. the population, we want to monitor. To this end, we derive a practical working view on Linked Data on the web by contrasting two popular views on Linked Data on the web: the Billion Triple Challenge dataset, which is a crawled dataset, and the LOD cloud[7], which is a visualisation of entries in CKAN (also known as the *datahub*), a registry for datasets. We decide for a hybrid approach that combines the two views. Next, we describe how we obtain a list of URIs to monitor: We sample the population, i. e. our working view on Linked Data on the web, and define a crawling scheme to expand our sample. Last, we describe and briefly validate our monitoring scheme.

4.6.1. How to Capture Linked Data on the Web?

Using our data collection, we want to enable researchers to investigate various aspects of the dynamics of Linked Data on the web. With Linked Data being only a set of practices

[7] `http://lod-cloud.net/`, visited on 2018-06-10.

to publish data on the web [17] to which publishers adhere to a varying extent, there is no yardstick or authority to tell which sources on the web the Linked Data community would consider in-scope or out-of-scope. Hence, the *population* we want to study does not have a clear definition that is practical for our purposes.

To get to a working definition of the population, we contrast two views on Linked Data on the web popular at the time of the design of the study: The Billion Triple Challenge dataset, specifically the 2011 edition, and the LOD cloud, represented by the underlying dataset from CKAN/DataHub available as of September 29, 2011.

BTC2011 – The Billion Triple Challenge Dataset

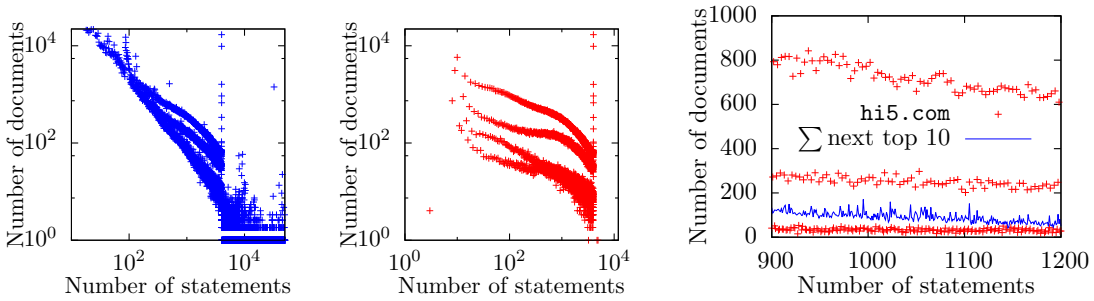
In the years 2008 to 2014, the Billion Triple Challenge to showcase applications that build on Linked Data has been performed at the International Semantic Web Conference (ISWC) series using a dataset crawled from the web [94, 95, 96, 97, 129]. In different years, the dataset has been composed by downloading RDF from the web and following links. This crawling approach to composition biases the dataset towards general-purpose and well-interlinked data providers. Conversely, data of providers in a niche application area with a less dense link structure –regardless of size and importance in that area–, may be under-represented in such a crawl.

For our analyses, we looked at the dataset of the 2011 edition of the Billion Triple Challenge [96] (short: BTC2011), the most recent dataset available at the time when we designed the Dynamic Linked Data Observatory. The dataset was crawled in May/June 2011 and contains 2.145×10^9 quadruples, i. e. RDF triples with the information resource that provided the triple in fourth position. These triples came from 7.411×10^6 information resources, specifically information resources that provide RDF in RDF/XML serialisation. The information resources reside on 791 pay-level domains (PLDs).

Definition 7 (Pay-Level Domain)**.** *The term PLD has been defined [147] as any domain that requires payment at a top-level domain or country-code top-level domain registrar.*

In practice, a pay-level domain is a direct sub-domain of a top-level domain (TLD) or a second-level country domain (ccSLD), e.g., `dbpedia.org`, `bbc.co.uk`. We prefer the notion of a pay-level domain since fully qualified domain names (FQDNs) overexaggerate the diversity of the data: for example, sites such as `livejournal.com` assign different subdomains to individual users (e.g., `danbri.livejournal.com`), leading to millions of FQDNs on one site, all under the control of one publisher: The BTC2011 dataset contained documents from 240 845 FQDNs, 233 553 of which were from the `livejournal.com` PLD. Henceforth, when we mention domain, we thus refer to a PLD (unless otherwise stated). The notion of the PLD has also been called "site" in earlier work [25] or "top private domain" in a Java library for URI processing[8]. Today, Mozilla maintains the "public suffix list"[9], a list of mappings to determine the PLD for a given FQDN.

[8] `http://github.com/google/guava`, visited on 2018-06-10.
[9] `https://publicsuffix.org/`, visited on 2018-06-10.



(a) Overall (b) `hi5.com` (c) `hi5.com` dominates

Figure 4.1.: Distribution of the number of statements in documents for the BTC2011 dataset (4.1a) overall and (4.1b) for `hi5.com`; as well as (4.1c) the periodicity of distribution of statements-per-document for `hi5.com` that causes the split tail in (4.1a) & (4.1b).

As the seed list for the crawl has been sampled from the dataset from the previous year, the dataset is hard to analyse by looking at the dataset's genesis. Hence, we look at what the dataset actually contains.

We found the dataset heavily dominated by few exporters that use the `foaf` vocabulary, hinting at social networks. We already discussed the dominance of `livejournal.com` when it comes to FQDNs. Another social network, `hi5.com` dominates the dataset when it comes to the ratio of statements per document: as observed for similar corpora (cf. Table A.1 in [117]) `hi5.com` has many documents, each with an average of over two thousand statements – an order of magnitude higher than most other domains. The dominance of `hi5.com` – and to a lesser extent similar sites like `livejournal.com` – shape the overall characteristics of the BTC2011 dataset. To illustrate one prominent such example, Figure 4.1a gives the distribution of statements per document in the BTC dataset on log/log scale, where one can observe a rough power-law(-esque) characteristic. However, there is an evident three-way split in the tail emerging at about 120 statements, and ending in an outlier spike at around 4 000 statements. By isolating the distribution of statements-per-document for `hi5.com` in Figure 4.1b, we see that it contributes to the large discrepancies in that interval. The stripes are caused by periodic patterns in the data, due to its uniform creation: on the `hi5.com` domain, RDF documents with a statement count of $10 + 4f$ are heavily favoured, where ten triples form the base of a user's description and four triples are assigned to each of f friends. Other lines are formed due to two optional fields (`foaf:surname`/`foaf:birthday`) in the user profile, giving a $9 + 4f$ and $8 + 4f$ periodicity line. An enforced ceiling of $f \leq 1,000$ friends explains the spike at (and around) 4 010. The other lines can be explained as follows: If a user on `hi5.com` does not state his birthday, there is no statement in the exported foaf file for birthday. Another reduction in statements is caused by the employed RDF/XML-parser, which drops statements that have invalid characters in their XML.

The core message here is that although BTC2011 offers a broad view of Linked Data

on the web, covering 791 domains, in absolute statement-count terms, the dataset is skewed by a few high-volume exporters, and in particular `hi5.com`. When deriving global statistics and views from BTC2011, the results say more about the code used to generate `hi5.com` profiles than the efforts of thousands of publishers[10]. In defence of the BTC2011 dataset, we note that such a dominance is a naturally-occurring phenomenon and is not limited to this dataset [52, 117].

Cloud2011 – The Linking Open Data Cloud Diagram

The cloud diagram from the Linking Open Data project, commonly dubbed as the "LOD Cloud", is a visualisation of Linked Data datasets and links. For a dataset to appear in the diagram, the dataset needs to be registered in a registry, the DataHub built on the CKAN software, specifically the group `lodgroup` therein[11] and to fulfil minimum requirements regarding certain self-reported numbers such as size and linkage with other datasets. These conditions for appearance bias this view on Linked Data on the web towards datasets whose providers put in the effort to sign up, and big datasets served on one domain. Conversely, small datasets published in a decentralised fashion, e. g. social network information published using the `foaf` vocabulary, or vocabularies themselves, which are highly relevant to Linked Data, often do not make it into the diagram.

For our analyses, we looked at the self-reported statistics as of September 29, 2011 (short: Cloud2011). At this time, there were datasets from 206 FQDNs or 133 PLDs. The reported triple count was more than one order of magnitude higher than for BTC2011. Most of the datasets claim to serve RDF/XML (85 %).

We found huge triple counts reported, but a quick check on the actual support of serialisation formats (vs. the reported support) indicates that the self-reported statistics and properties have to be taken with a grain of salt.

BTC2011 vs. Cloud2011

We next contrasted the two views on Linked Data on the web. Overall, both contain data from 854 PLDs, with BTC2011 covering 791 domains (∼92.6 %), Cloud2011 covering 133 domains (∼15.6 %), and the intersection of both covering 70 domains (∼8.2 % overall; ∼8.8 % of BTC2011; ∼52.6 % of Cloud2011). Cloud2011 reports a total of 28.4×10^9 billion triples, whereas the BTC (an incomplete crawl) accounts for 2.1×10^9 billion quadruples (∼7.4 % of Cloud2011). However, only 384.3×10^6 million quadruples in the BTC2011 dataset (∼17.9 %) come from PLDs mentioned in the extracted Cloud2011 metadata. In Table 4.1, we present the BTC2011 and Cloud2011 statement counts side-by-side. We can observe that a large number of high-volume BTC2011 domains are not mentioned on Cloud2011, where the datasets in question may not publish enough RDF data to be eligible by Cloud2011, or may not follow Linked Data principles or have enough external links, or may not have self-reported.

[10] Furthermore, `hi5.com` is not even a prominent domain on Linked Data on the web in terms of being linked, and was ranked 179/778 domains in a PageRank analysis of a similar corpus, see

Table 4.1.: Statement counts for top-25 PLDs in the BTC2011 with corresponding reported triple count in Cloud2011 (left), and top-25 PLDs in Cloud2011 with BTC2011 quad count (right)

No.	Top-25 BTC2011			Top-25 Cloud2011		
	PLD	*BTC2011*	*Cloud2011*	*PLD*	*Cloud2011*	*BTC2011*
1	hi5.com	1 371 854 358	—	rpi.edu	9 803 140 000	900 464
2	livejournal.com	169 863 721	—	linkedgeodata.org	3 000 000 000	—
3	tfri.gov.tw	153 300 321	23 015 257	legislation.gov.uk	1 900 000 000	31 990 934
4	scinets.org	56 075 080	—	wright.edu	1 730 284 735	5
5	ontologycentral.com	55 124 003	122 000 000	concordia.ca	1 500 000 000	—
6	rdfize.com	36 154 381	—	data.gov.uk	1 336 594 576	13 302 277
7	legislation.gov.uk	31 990 934	1 900 000 000	dbpedia.org	1 204 000 000	25 776 027
8	identi.ca	30 429 795	—	rdfabout.com	1 017 648 918	—
9	bibsonomy.org	28 670 581	—	dbtune.org	888 089 845	1 634 891
10	dbpedia.org	25 776 027	1 204 000 000	uniprot.org	786 342 579	4 004 440
11	freebase.com	25 488 720	337 203 427	unime.it	586 000 000	—
12	opera.com	23 994 423	—	uriburner.com	486 089 121	—
13	bio2rdf.org	20 168 230	72 585 132	openlibrary.org	400 000 000	25 396
14	archiplanet.org	13 394 199	—	sudoc.fr	350 000 000	—
15	data.gov.uk	13 302 277	1 336 594 576	freebase.com	337 203 427	25 488 720
16	loc.gov	7 176 812	24 151 586	fu-berlin.de	247 527 498	5 658 444
17	vu.nl	6 106 366	14 948 788	dataincubator.org	205 880 247	3 695 950
18	bbc.co.uk	5 984 102	80 023 861	viaf.org	200 000 000	—
19	rambler.ru	5 773 293	—	europeana.eu	185 000 000	—
20	fu-berlin.de	5 658 444	247 527 498	moreways.net	160 000 000	—
21	uniprot.org	4 004 440	786 342 579	rkbexplorer.com	134 543 526	220
22	dataincubator.org	3 695 950	205 880 247	ontologycentral.com	122 000 000	55 124 003
23	zitgist.com	3 446 077	60 000 000	opencorporates.com	100 000 000	—
24	daml.org	3 135 225	—	uberblic.org	100 000 000	—
25	mybloglog.com	2 952 925	—	geonames.org	93 896 732	458 490

Perhaps more surprisingly however, we note major discrepancies in terms of the catchment of BTC2011 statements versus Cloud2011 metadata. Given that BTC2011 can only *sample* larger domains, a lower statement count is to be expected in many cases: however, some of the largest Cloud2011 domains do not appear at all. Reasons can be found through analysis of the BTC2011's publicly available access log [96]. In Table 4.2, we present reasons for the top-10 highest-volume Cloud2011 data providers not appearing in the BTC2011 dataset (i. e. providers appearing with "—" on the right-hand side of Table 4.1). Robots indicates that crawling was prohibited by `robots.txt` exclusions; Http-401 and Http-502 indicate the result of lookups for URIs on that domain; Mime indicates that the content on the domain did not return `application/rdf+xml` used as a heuristic in the BTC2011 crawl to filter non-RDF/XML content; Unreachable indicates that no lookups were attempted on URIs from that domain; Other refers solely to `europeana.eu`, which redirected all requests to their home page.

Concluding, we found that both views have strengths and weaknesses, and observe major divergence with respect to coverage between the two. As our aim is to provide a general-purpose dataset, which should be interesting to a wide range of parties, we decided to take a view on linked data that unifies the best of both BTC2011 and Cloud2011.

Table 4.2.: Reasons for largest ten PLDs in Cloud2011 not appearing in BTC2011.

PLD	Robots	Http-401	Http-502	Mime	Unreachable	Other
`linkedgeodata.org`			X	X		
`concordia.ca`				X		
`rdfabout.com`				X		
`unime.it`					X	
`uriburner.com`	X					
`sudoc.fr`					X	
`viaf.org`				X		
`europeana.eu`						X
`moreways.net`					X	
`uberblic.org`		X				

4.6.2. Sampling Technique

Due to the size of Linked Data on the web and the need for frequent snapshots, sampling is necessary to create an appropriate collection of URIs that can be processed and monitored

`http://aidanhogan.com/ldstudy/table21.html`, visited on 2018-06-10.

[11] `http://ckan.org/`, now at `https://old.datahub.io/dataset?tags=lod`, visited on 2018-06-10.

under the given time, hardware and bandwidth constraints. The goal of our sampling method is thus two-fold: to select a set of URIs that (i) capture a wide cross-section of domains and (ii) can be monitored in a reasonable time given our resources and in a polite fashion. Given the previous discussion, we wish to combine the BTC2011 (crawled) and Cloud2011 (metadata) perspectives when defining our seed-list.

Before we continue to describe the sampling methodology we choose, it is worthwhile to first remark upon sampling methods used in other dynamicity studies of the Web.

Published Sampling Techniques

There are several published works that present sampling techniques in order to create a corpus of Web documents that can be monitored over time. Having studied a variety of major works, we could not find common agreement on a suitable sampling technique for such purposes. The targeted use-cases and research questions directly affect the domain and number of URIs, as well as the monitoring frequency and time frame.

Grimes and O'Brien [88] studied the dynamics of highly frequently changing pages and prepared their seed list accordingly. From a crawl of the search engine provider Google, they selected hosts according to the number of URLs they had. From that, they sampled "a number of URLs" and crawled them twice a day and took the most dynamic of them. From those URLs they chose those with at least 500 successful fetches, which allows for assumptions on the reliability of the provider.

Fetterly et al. [72] studied the dynamics of changes within pages and the Web in general. In order to allow for general statements, they took the Yahoo! homepage as a source of URLs covering many subjects and types of providers of pages. From there, they crawled and took a random sample of the URLs.

In their development of an estimator for change frequency of web pages, Cho and Garcia-Molina [44] crawled the Web for the validation of their estimator, so they also need a broad coverage of various pages. They started off with a snapshot of 2.5×10^7 web pages they already had, from which they selected the most popular 270 sites (cf. PLD, definition 7) according to their in-links. From each site, they downloaded 3 000 pages, which they downloaded every day for their analysis.

We observe that where researchers employed sampling, the exact numbers employed in the sampling are not described as calculated for the special needs of their analyses. Instead, the numbers seem like a result of the trade-off between broad coverage and resource consumption.

Our Sampling Technique

We can conclude that existing sampling methods select URIs from crawled documents, either randomly, because of specific characteristics (e. g. dynamic or highly ranked), or to ensure an even spread across different hosts. Thus, we decide to use a combination of these three methods to generate our list of URIs.

Similarly to the works in literature, we take into account two factors when sampling: coverage of different hosts and bias for certain characteristics. To compile our initial

seed list, we took all 220 example URIs in the registry for Cloud2011 (for coverage, similar to [72]). To this list, we added the 220 most popular URIs from the BTC2011 dataset, determined by a PageRank analysis [85] (for the characteristics popularity and interlinkage, similar to [44]). Thus, we also include popular vocabularies, which are amongst the most linked Linked Data documents. These 440 core URIs contain 137 PLDs, 120 from the Cloud2011 examples and 37 from the most popular BTC2011 URIs. To build a corpus of a meaningful size, we extend these 440 URIs by crawling. From the data thus downloaded, we obtain the final set of URIs to monitor.

The result is then our best-effort compromise to achieve representative snapshots of Linked Data that (i) take into account both views on Linked Data by including Cloud2011 and BTC2011 URIs in the core seed list, (ii) extend beyond the core seed list in a defined manner, and (iii) do not exceed our crawling resources.

4.6.3. Crawling Setup

From our observations, the specifics of the crawling configuration have a significant effect on the data. For instance, the reported support of serialisation formats indicates that by restricting ourselves to RDF/XML, like BTC2011, we may miss out on 15 % of data providers from Cloud2011. Thus, we provide details and justifications on our crawling configuration: Our implementation is based on two open source Java projects: (1) LDSpider[12], a multi-threaded crawling framework for Linked Data [124], and (2) any23[13], a parsing framework for various RDF syntaxes. The experiments are intended to run on a dedicated, single-core 2.2 GHz Opteron x86-64 with 4 GB of RAM on a university network. We use the following configuration:

All RDF syntaxes: we want to consider all standard serialisation formats for RDF as supported by LDSpider or any23, including RDF/XML, Turtle, N-Triples, and RDFa. Moreover, we do not want to penalise providers for incorrectly implementing content-negotiation or falsely suggesting formats in file extensions, and thus do not pre-filter on `Content-type` header information or URI substrings. For RDFa, where RDF extracted from HTML can either be meta data from HTML headers, e. g. on stylesheets and icons, or data from the HTML body, which we consider the "intended" data, we are rather interested in the latter and filter out the former "accidental" trivial triples.

Threads: to make the most of our hardware in presence of slow sources and parsing, we use a multi-threaded crawler configuration. In previous work [117], we found 64 threads to offer the best trade-off between parallelism and CPU/disk contention.

Timeouts: Working on the web, we have to deal with network outages and misconfigured hard- and software. Hence, we set a timeout on connections and sockets to 120 s. The timeout is deliberately high to ensure reproducibility.

[12] `http://github.com/ldspider/ldspider`, visited on 2018-06-10.
[13] `http://any23.org/`, visited on 2018-06-10.

Links: When crawling, we have to decide which links we want to follow (cf. the 4th Linked Data Principle [17]). We follow all links, instead of e. g. restricting us to `owl:sameAs` or `rdfs:seeAlso` links.

Breadth-first: As crawling strategy, we chose breadth-first crawling, which, given a diverse seed-list, should result in a diverse data-set [117].

Redirects: On the web of Linked Data, redirects, i. e. HTTP responses with `301`, `302`, `303` or `307` status code, are often used to express the difference between non-information resources and information resources[14]. Hence, we do not consider redirects as links, but follow the redirects immediately within the same crawling hop, until we get a response with a successful status code (`2xx`) or an error (status code $\geq$`400`, network error, or we hit a cycle or path length limit).

Per-domain Queue: To implement polite crawling in an efficient fashion, our crawler maintains an individual queue per PLD. By polling these queues in a round-robin fashion, we can ensure a maximal delay between two accesses to a PLD while making high use of our crawler's capacity.

Priority Queue: To cover the most important URIs per PLD when performing incomplete crawls, we order the URIs in the per-PLD queues according to the number of in-links we encountered so far.

Politeness Policy: For polite crawling, we implemented a 2 s delay between two requests to a PLD. Moreover, we honour `robots.txt` exclusions.

When crawling a scale-free network like the web in a polite fashion, the phenomenon of "PLD-starvation" can occur [117], which we want to counter: The number of active PLDs (i. e. PLDs with still URIs in the queue) times the politeness delay results in the minimum amount of time to complete a round of the per-PLD queues. In the worst case, only one PLD is active, which keeps the crawling hop unfinished until all URIs from that PLD are downloaded in a polite fashion. This "starvation" results in a very low utilisation of crawling resources. On the other hand, in the frontier, i. e. the list of URIs for the next hop, there are a lot of URIs which could raise the utilisation. Hence, we employ a termination criterion for the hops: If (1) the list of seed URIs has been downloaded completely, *and* (2) all redirects have been followed *and* (3) the number of active PLDs drops below the number of threads, we end the current hop and move the remaining URIs to the frontier. The priority queues ensure that we cover the most important URIs nevertheless.

4.6.4. Compiling the Seed List

Starting from our list of 440 core URIs, we wish to expand a 2-hop crawl using the outlined framework, from which we will extract the final seed list of URIs to monitor in our observation framework. However, due to the unpredictability/non-determinism

[14] Discussed under the headline "httpRange-14" at `http://www.w3.org/2001/tag/group/track/issues/14`, visited on 2018-06-10.

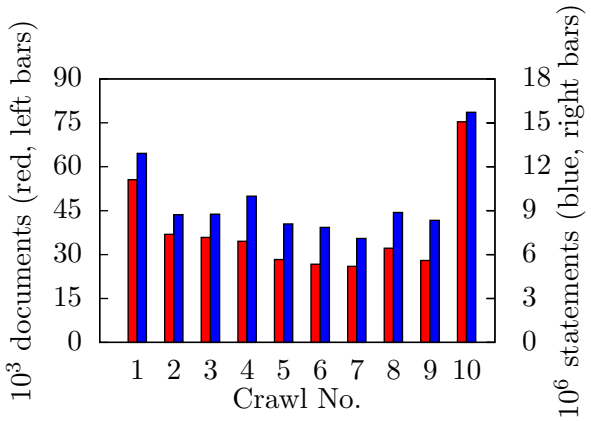


Figure 4.2.: Number of statements and documents per crawl experiment.

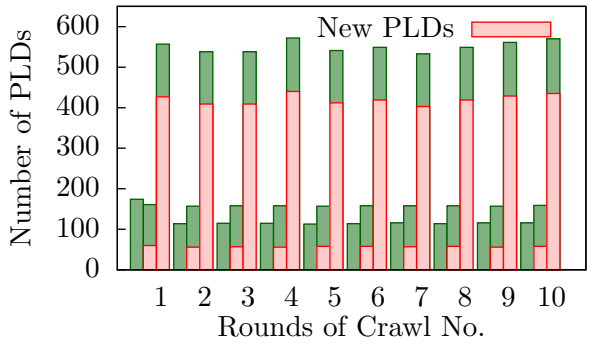


Figure 4.3.: Number of PLDs per round per crawl experiment.

of remote connections, we want to ensure a maximal coverage of the documents in this neighbourhood. Along these lines, we repeated ten complete 2-hop crawls from our core URI list.

With respect to the non-determinism, Figure 4.2 shows for each round the number of documents (left bars on y-axis) and the number of statements (right bars on y-axis). We can observe that two crawls (crawl number 1 and 10) have a significantly higher number of statements compared to the other crawls. One reason for this large discrepancy relates to the `identi.ca` domain, where a URI (referring to Tim Berners-Lee's account; a highly ranked document in the BTC2011 dataset) in the seed round of crawls 1 and 10 offered a rich set of links within that domain, whereas the lookup failed in the other crawls, cutting off the crawler's path in that domain: for example, in the first crawl, `identi.ca` accounted for 1.5×10^6 million statements, whereas in crawl 2, the domain accounted for 1.7×10^4 statements. Such examples illustrate the inherent non-determinism of crawling.

In Figure 4.3, we show for each crawl the number of visited PLDs per round together with the number of new PLDs per round with respect to the previous round. The left bar for each crawl represents Round 0, the middle bar Round 1, and the right bar Round 2.

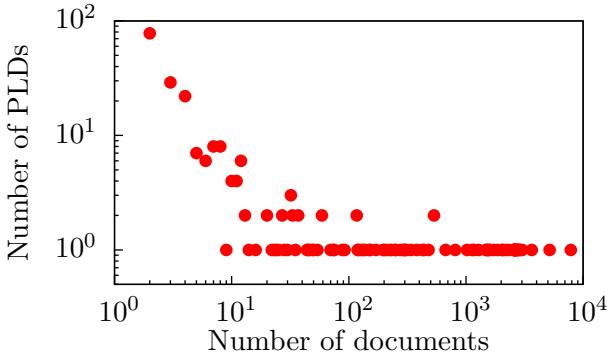


Figure 4.4.: Distribution of the number of documents per PLD.

Table 4.3.: Top 10 PLDs based on the number of URIs.

No.	PLD	URIs
1	`gesis.org`	7 850
2	`chem2bio2rdf.org`	5 180
3	`dbpedia.org`	3 643
4	`freebase.com`	3 026
5	`fer.hr`	2 902
6	`loc.gov`	2 784
7	`concordia.ca`	2 784
8	`dbtune.org`	2 767
9	`fu-berlin.de`	2 689
10	`semantictweet.com`	2 681

We can observe that the relative level of domains across the crawls is much more stable when compared with the number of documents (cf. Figure 4.2). Across rounds, the graph shows an average ∼1.3× increase of active PLDs between Rounds 0 and 1, and an increase of ∼3.4× between Rounds 1 and 2. Further, we observe that about 30 % of the PLDs in Round 1 are new compared to the previous round and roughly 70 % of the PLDs in Round 2 are not visited by the crawler in the rounds before.

Given the non-deterministic coverage of documents, to ensure comprehensive coverage of URIs in the 2-hop neighbourhood, we take the union of URIs that dereferenced to RDF content, resulting in a total set of 95 737 URIs spanning 652 domains, giving an average of 146.8 dereferenceable URIs per domain. Figure 4.4 shows in log/log scale the distribution of the number of PLDs (y-axis) against the number of URIs in the union list (x-axis); we observe a typical power-law-ish distribution and see that 379 PLDs (∼58.1 %) have one URI in the list, 78 PLDs (∼12.0 %) have two URIs, and so forth. In addition, Table 4.3 lists the number of URIs for the top-10 PLDs in the set (represented by the ten rightmost dots in Figure 4.4).

4.6.5. Monitoring Scheme

For the monitoring scheme, we have to decide along two dimensions: Crawling set-up and interval. Both are subject to resource constraints.

For the crawling set-up, resource constraints cause us to limit the number of URIs we download. The spectrum spans between downloading the same list of URIs every time and downloading a different list of URIs every time, where in the latter case the list of URIs is determined by the previous crawls or the initial URI list. The former end of the spectrum allows for studying the evolution of the list of URIs in a **contiguous** fashion and is widely followed in literature [28, 29, 43, 44]. We follow the hybrid approach of [163] by downloading the same list of URIs every time and by also performing a crawl from this URI list to incorporate new sources. This way, we want to balance **contiguousness** and **adaptiveness** in our set-up. Yet, to answer the research questions of Section 4.2, we require a contiguous dataset. Hence, we disregard the crawled part in this thesis.

For the monitoring interval, resource constraints cause us to set a frequency for monitoring. The frequency in previous studies based on practical considerations, e. g. weekly [28, 29, 43, 44], daily [163, 72], or even hourly [88]. Given our resources, we decided for a weekly interval.

4.6.6. Validation of the Feasibility the Proposed Set-up

To validate our set-up, we performed a download of the 95 737 seed URIs and monitored the crawler's behaviour. The download took 5 h 55 min, during which we downloaded about 16×10^6 statements from about 80×10^3 documents. In Figure 4.5, we show the number of requests performed per crawl hour. During the first hours, we observe the highest rate of about 70×10^3 requests per hour. After that, the rate drops significantly due to PLD starvation: In the last hour, only about 1.2×10^3 requests have been made. Overall, we observe that, including a safety margin, we can process the seed list within a 10 hours' interval.

4.7. High-Level Statistics

We have performed this monitoring on a weekly basis since 2012/05/06, yielding 299 weekly snapshots at the time of writing, spanning exactly 6 years. Each snapshot consists of the content retrieved for the core kernel URIs (following redirects), the content of the expanded crawl, a set of redirects, and access logs for URIs that were looked up. In this thesis, we focus on the kernel. Table 4.4 enumerates the average and total amount of data retrieved over the 299 weeks for the kernel documents. Of the 95 737 kernel URIs, an average of 64 243 URIs could be dereferenced successfully: though all URIs were deemed to dereference to RDF during sampling, some now fail to dereference. The number of unique URIs dereferencing successfully in at least one snapshot was 95 567 and the analogous figure for domains was 649 (vs. 652 for the source URIs). In terms of the diversity of the kernel, the documents in each snapshot came from an average of 528 PLDs.

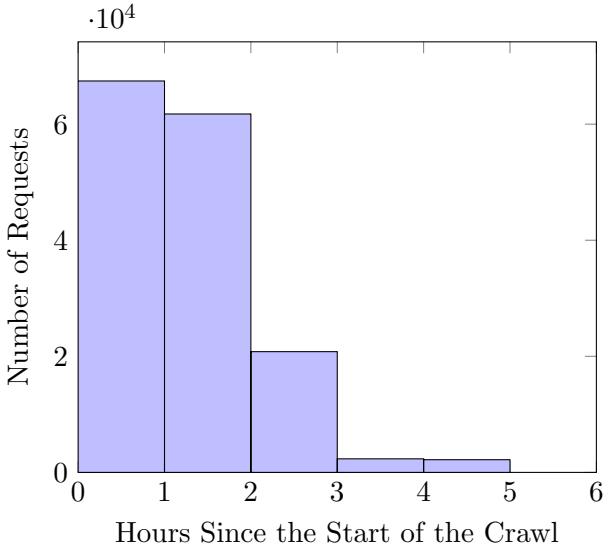


Figure 4.5.: Number of requests made over time.

Table 4.4.: Overall statistics across all 299 snapshots.

Mean pay-level domains	527.7	±59.5
Mean seed URIs dereferenced	64 243.3	±11 121.6

4.8. Methods to Describe and Analyse the Dynamics of Linked Data

There is no standard way to analyse dynamic Linked Data. For instance, for the analysis of dynamic Linked Data in the set-up and after the first half year, we used a multi-stage pipeline controlled on the Linux command line, where the components were mostly written in Java. In a similar fashion, other researchers conducted their work, e. g. the code to a paper by Dividino et al. can be found online[15]. While the approach to use a general-purpose imperative programming language leaves room for manual optimisation of the code, to gain understanding how such code performs the analyses is difficult. Hence, the validation of the analyses is hard to carry out.

Motivating Example. *In [128], we reported the number of overall documents ever appearing in the first half year of the Dynamic Linked Data Observatory dataset, which monitors a set of* 95 737 *seed URIs, to be* 86 696*. To be precise, this was the number of information resources ever encountered in any of the* 29 *snapshots available at that time and determined by counting the contexts of quads. When considering e. g.* 171 *snapshots, this number rises to about* 106 105 *documents, more than seed URIs. This mismatch points us towards that we may have to re-ask our question and take into account the dereferencing process: If we instead consider the number of seed URIs that dereferenced to information resources, the number for* 29 *snapshots is* 92 712*. To further put this number into context, we note that for the first snapshot,* 67 107 *seed URIs were redirected, i. e. had non-trivial dereferencing, and* 77 958 *could be dereferenced successfully.*

To specify the analyses in a way that is easier to validate and that requires less programming effort, we therefore want to investigate a declarative approach to analyse dynamic Linked Data. The Linked Data principles mention the declarative SPARQL language for querying RDF data. Hence, we next investigate the applicability of SPARQL and RDF to analyse dynamic Linked Data. Specifically, we want to model the snapshots from the Dynamic Linked Data Observatory in RDF, and use queries in the declarative SPARQL language for the analyses. In this declarative approach, imperative code is then only required to produce data according to the model.

Hence, in this section we ask the research question: Can we use the Semantic Technologies of SPARQL and RDF for the analyses of dynamic Linked Data?

4.8.1. Steps to Analyse Dynamic Linked Data

To analyse dynamic Linked Data based on weekly snapshots requires a three-step processing pipeline:

1. Extraction of the relevant data from the raw data and log files generated weekly into a model

2. Preprocessing of the data, e. g. application of a uniform handling of missing values

[15] `https://github.com/dividino/LODCacheUpdateStrategyEvaluation`, visited on 2018-06-10.

3. Analysis of the data

With the aim of reducing the amount of imperative code, we want to cover step 2 and 3 declaratively using Semantic Technologies. For step 1, we have to rely on imperative code, as there no declarative approach to parse all the log files of the crawler employed in the Dynamic Linked Data Observatory, LDSpider.

As basis for our considerations, we provide an ontology in RDFS to represent information both at the physical and logical levels of dynamic Linked Data. We then can present our processing pipeline:

1. Extraction of the relevant data from the raw data and log files generated weekly into an RDFS model using imperative code
2. Preprocessing of the data, e. g. application of a uniform handling of missing values using SPARQL UPDATE queries
3. Analysis of the data using SPARQL SELECT queries

We now present first the model and then the steps of the processing pipeline.

4.8.2. Modelling Dynamic Linked Data in RDFS

Using our model, we want to allow for analysing both the data access and the data itself in an integrated fashion. The basis for our considerations is an approach that periodically creates snapshots of Linked Data by dereferencing a set of URIs. Hence, central to our ontology is an observation of one URI at one point in time. We show our proposed ontology in Figure 4.6.

To represent the details about the **physical level** of Dynamic Linked Data, our model captures in an RDF list the HTTP requests following redirects where dereferencing a seed URI. RDF lists are closed and ordered, both features are desired for the requests, as the number of requests that has been made is determined by what happened, and the order of the request is determined by the redirects. Nonetheless, we highlight that to traverse the elements in RDF lists in queries, it is necessary to use SPARQL property paths, which may be rather expensive to evaluate depending on the assumed semantics [7]. To allow for querying the information resource of a seed URI, the model includes the `:hasLastResponse` property. In this way, it is not necessary to traverse the entire list of requests thus reducing the complexity of queries that only require data about the information resource associated with a seed URI. For each request, the model represents the obtained response which contains the status code and the response body (if applicable).

The **logical level** of Dynamic Linked Data corresponds to the RDF Graphs obtained in the body of a response. To associate these graphs with the corresponding response, the proposed model represent the triples in the RDF Graphs as an RDF list of reified triples. Reified triples allow to make statements about triples without logically asserting the triples. In this context, reified statements allows for referring to RDF Graphs at different points in time. Another option would have been named graphs for each information

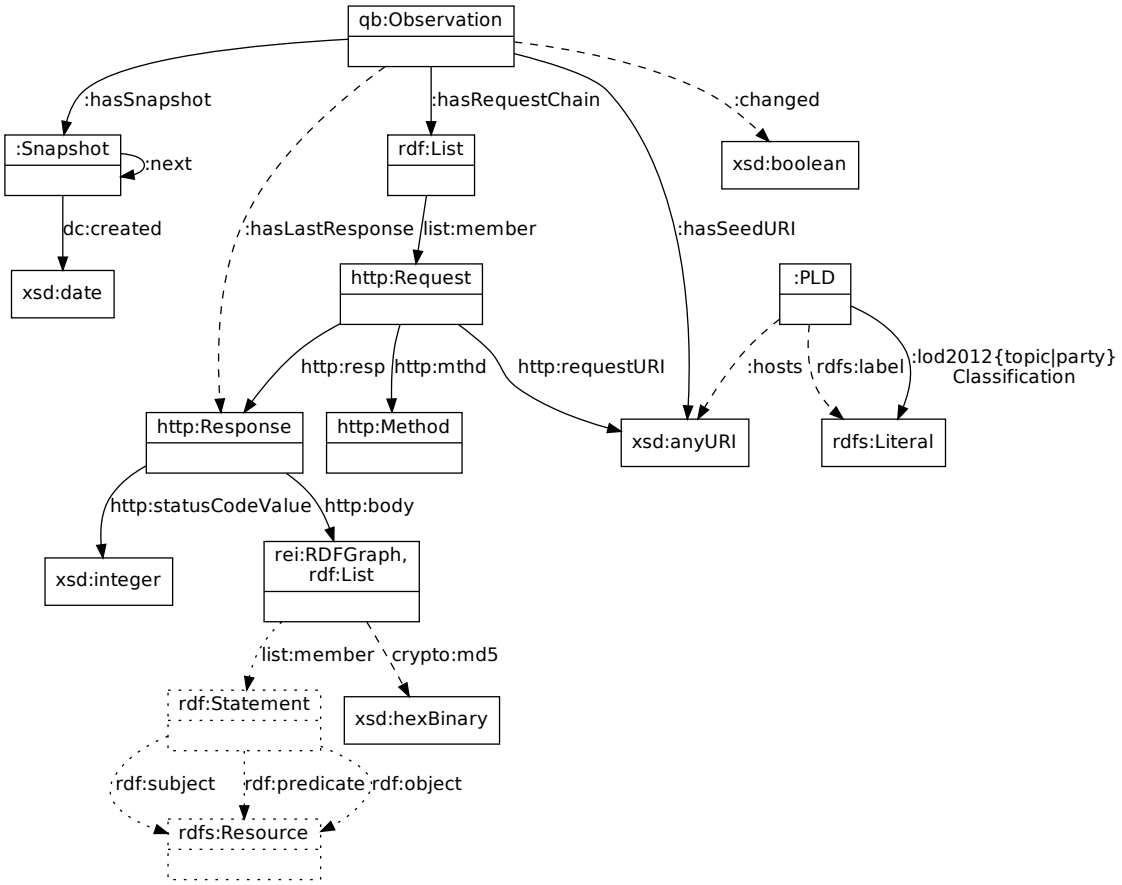


Figure 4.6.: Our proposed ontology to describe dynamic Linked Data in RDFS. We note that the dashed information can be inferred. We did not use the dotted information to produce the figures in this thesis.
The depiction uses the UML class diagram with the following correspondence from UML to RDFS: UML classes depict `rdfs:Class`es and UML associations depict `rdfs:domain` and `rdfs:range` of an `rdf:Property`. We use `list:member` associations to state the `rdfs:Class` of the members of an `rdf:List`.

resource at each point in time. Nonetheless, this option has several drawbacks. First, we would have to name each named graph in `FROM` clauses of analysis queries. With about 300 snapshots of about 95 737 URIs, we would have about 30×10^9 `FROM` clauses in the queries, which is rather lengthy. Second, to try different triple stores, whose treatment of named graph varies if they are supported (some treat the triples in the named graphs as asserted in the default graph), we want to rely on common features independently from the triple store. To efficiently find out whether a HTTP request to a given URI yielded two different graphs at two different points in time, we have to check the graphs for isomorphism. In the presence of blank nodes, such checks are difficult in SPARQL. Hence, we replace the blank nodes by URIs using the hash-based skolemisation approach described by Hogan in [115]. Thus, we can check using RDF term equality whether a triple with a blank node from the graph derived at one point in time is the same as a triple with a blank node from another point in time. We model the corresponding hashes using the `crypto:md5` predicate. To materialise the expensive computation of changes, we use the `:changed` predicate.

To allow for analyses on the level of PLDs, we model the PLDs and the URIs from the seed URI the PLD hosts, see the predicate `:hosts` in Figure 4.6. Moreover, to perform analyses based on the classifications of the Linking Open Data cloud of 2012, we also model the classifications of PLDs according to topic and party, see the predicates `:lod2012topicClassification` and `:lod2012partyClassification` in Figure 4.6.

4.8.3. A Processing Pipeline to Extract Data According to Our Model

The processing pipeline includes the handling of the data produced by the tools to create snapshots of Linked Data. In our approach, the pipeline obtains relevant information about the dynamics of the crawled Linked Data that is scattered in different LDSpider outputs into one RDF Graph. While LDSpider uses a de-facto standard for some of the log files, e. g. the format of the popular Squid HTTP cache[16], which could be declaratively extracted e. g. using regular expressions to extract the data into comma-separated text, and to use Tarql[17] to turn the comma-separated text into RDF, not all the data required is in such log files. For instance, errors on the networking level such as timeouts do not have a standardised HTTP error code, and hence do not appear in those log files.

We give an overview of the processing pipeline in Figure 4.7. The proposed pipeline includes two major components, which extract data per-snapshot, see Figure 4.7a: one to process the log files on the physical level of Linked Data[18], and another to process the quads on the logical level of Linked Data[19]. Moreover, two minor components provide links between the snapshots and to information about the PLDs of the URIs in the seed list. The minor components have only to be run once, see Figure 4.7b.

To integrate the data of the two major components using RDF merge [109], both mint URIs for the HTTP responses retrieved during the dereferencing process in the

[16] `http://www.squid-cache.org/`, visited on 2018-06-10.
[17] `http://tarql.github.io/`, visited on 2018-06-10.
[18] `http://github.com/kaefer3000/dyldo-http2rdf`, visited on 2018-06-10.
[19] `http://github.com/kaefer3000/dyldo2qb`, visited on 2018-06-10.

Table 4.5.: Assigning exceptions to HTTP status code classes.

Exception	Status Code Class	Rationale
SSL exceptions	5xx	SSL mis-configured on server
Connection pool timeout	5xx	Caused by invalid HTTP responses
Socket timeout	5xx	Server mis-configured
Socket exception	5xx	Server suddenly closed connection
URI syntax exception	5xx	Caused by server-provided redirects
No HTTP response	6xx	Unclear whose network is faulty
No route to host	6xx	
Unknown host	6xx	
Connection timeout	6xx	

same way. Then, the URIs generated from the last HTTP response in an HTTP request chain matches the URIs generated from the contexts in the quads of the snapshot. The data merged from all, major and minor, components follows the model presented in Figure 4.6. The meta data processing code adds custom HTTP status codes for URIs whose dereferencing yields non-HTTP errors such as networking errors. We assigned network errors to the HTTP status code classes whether we consider them sever or networking errors. For the networking errors, we added a custom status code class (6xx) to the standard status code classes. Those custom HTTP status codes allow us to model and query both HTTP errors and networking errors in a uniform fashion. We list the assignment of non-HTTP errors to status code class in Table 4.5 providing a rationale four assignment.

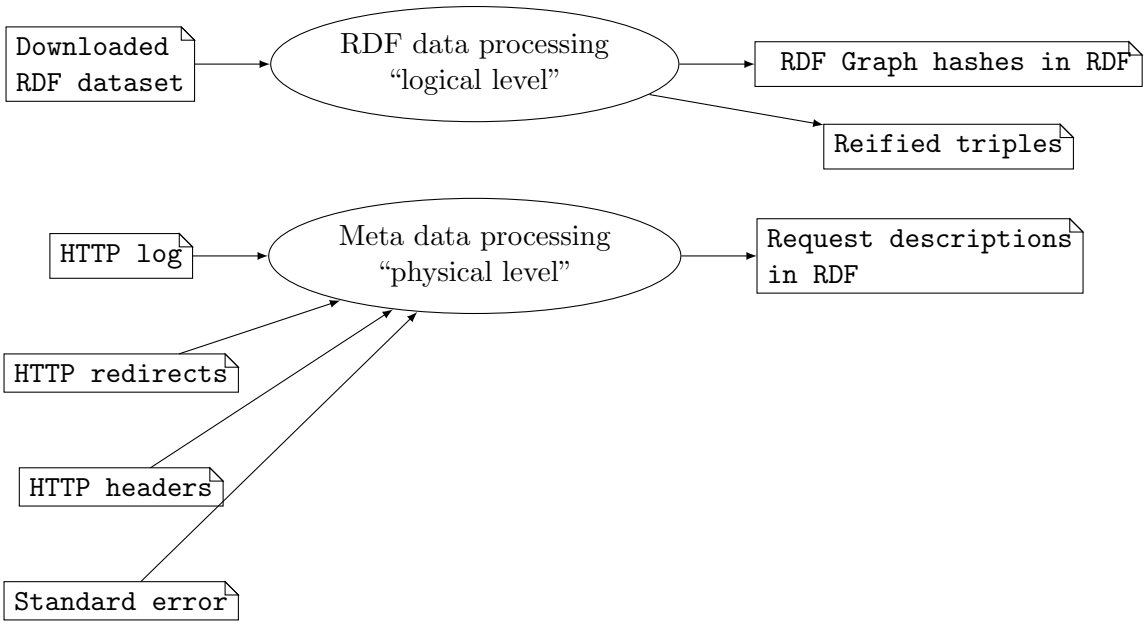
4.8.4. Declaratively Specifying Preprocessing using SPARQL UPDATE

We next give two examples for specifying preprocessing steps using SPARQL UPDATE queries.

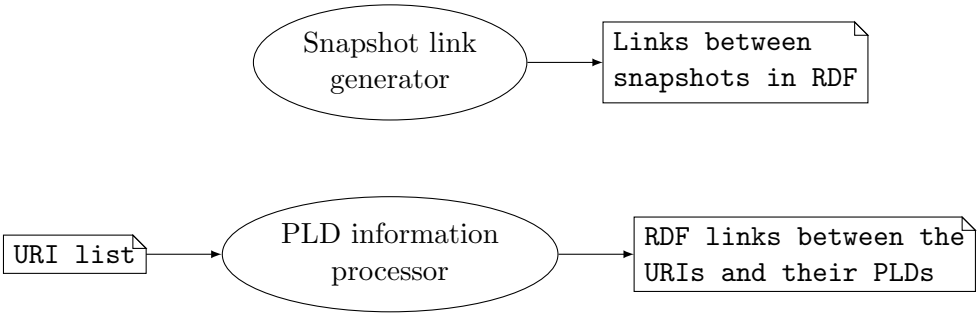
The Semantics of Unavailable Sources

One step is the definition of the semantics of offline sources. For instance, if the request to a URI returned successfully for two points in time with the same data, but unsuccessfully for the point in time in between, we have to define whether this behaviour constitutes a change. In this thesis, we do not consider unsuccessful responses as a change and specify this semantics using a SPARQL UPDATE query, see Appendix A.1.1. The query consists in two SPARQL INSERT queries:

Forward Propagation The first query checks for a given URI, whether there is an observation with an HTTP body for the last response followed by an observation without an HTTP body for the last response. For such pairs of last responses, the HTTP body is propagated from the last response with a HTTP body to the last



(a) The processing pipeline per snapshot. The input (on the left) is produced by a crawler. Both the physical and the logical level, and data from different snapshots can be processed in parallel.



(b) The processing pipeline to be run once.

Figure 4.7.: The proposed processing pipeline to produce data in our model of Figure 4.6 to describe dynamic Linked Data in RDF.

response without an HTTP body. In the presence of redirects, multiple URIs can redirect to the same last response. Thus, an unavailable source can get multiple HTTP bodies assigned using this propagation method. This result of multiple HTTP bodies is desired – if a source is online again, we have to check for all seed URIs for which the source was the last response, whether there was a change, hence all previous HTTP bodies are candidates for checks. Unfortunately, this adds complexity to the analysis queries. Term equality of the hashes of responses is not sufficient in the case of multiple HTTP bodies. Instead, we have to use `FILTER NOT EXISTS`, which complicates the queries.

Backward Propagation While the first query works forward, we also want to consider sources that only appeared in the Dynamic Linked Data Observatory after some time. For those sources, we have to propagate the first-observed HTTP body backward. Hence, the query checks for an observation of a seed URI whether there was ever a body downloaded before in a last response, and if not, the query propagates the HTTP body to the last response in the previous observation of the seed URI.

The query needs to be executed until no further data is added.

As the backward propagation involves a SPARQL property path to check whether there was a body ever before, which is computationally expensive, we can run the forward propagation query first until no further data is added, and then run a re-written version of the backward propagation query that contains no SPARQL property path (the check whether there was ever a body downloaded before is then superfluous).

Materialising Intermediary Results

A recurrent query part when investigating the dynamics of Linked Data is the check whether between two points in time, a URI dereferenced to the same or to different RDF Graphs. When working with our proposed model, this check requires the evaluation of 11 triple patterns and a filter in SPARQL. We can reduce this number to 5 triple patterns and no filter by pre-computing results: We evaluate a SPARQL UPDATE query with the 11 triple patterns and a `BIND` instead of the `FILTER` in the `WHERE` clause and add triples with `:changed` predicate (see Figure 4.6) using a corresponding `INSERT` clause. Then, we can use the `:changed` predicate in our analysis queries.

4.8.5. Declaratively Specifying Analyses using SPARQL

We provide queries to determine the high-level statistics (Hx) of Section 4.7 in Appendix A.2. Moreover, we provide queries to create the data for the first six figures (Qx) and Table 2 (T2) of our paper [128] in Appendix A.3. We list the features of the SPARQL language used by each query in Table 4.6.

Table 4.6.: SPARQL features used by different queries.

Query	No. TP	No. AGG	No. STR	No. SQ	No. TP in EXI	No. PP
H1	4	1	—	—	—	—
H2	5	1	—	—	—	—
H3	3	1	—	—	—	—
H4	4	1	—	—	—	—
Q1	4	2	—	1	—	—
Q2	5	1	2	—	—	—
Q3	3	1	—	1	4	1
Q4	3	1	—	2	4	1
Q5	10	1	—	—	1	—
Q6	5	8	—	1	6	—
T2	8	15	—	3	6	—

Abbreviations: TP – Triple Patterns, AGG – Aggregations, STR – String Manipulations, SQ – Subqueries, EXI – Exists, PP – Property Paths

4.8.6. Evaluation of the Declarative Approach to Analyse Dynamic Linked Data

We evaluate the use of Semantic Technologies for analysing dynamic Linked Data. First, we look at the RDFS model and SPARQL queries, then at data loading and execution. Specifically, we look at the queries on the physical level of [128] and five snapshots from the Dynamic Linked Data Observatory.

RDFS Model and SPARQL Queries

We gave a model of dynamic Linked Data in RDFS in Figure 4.6. RDFS allowed us to model all relevant aspects of dynamic Linked Data for the queries of [128]. While we gave the ontology in RDFS, we do not make use of RDFS entailment. For instance, the use of `rdfs:domain` and `rdfs:range` was only for documentation purposes.

The expressivity of SPARQL is sufficient to give queries for the analyses on the physical level of [128] (see Appendix A.3) and the required preprocessing (see Appendix A.1). In contrast to the imperative code we used before for the analyses, which spans thousands of lines of code, the queries are fairly concise. We did not investigate the logical level for scalability reasons, see below.

Initial Performance Experiments

We investigated queries for the first (Q1), the second (Q2), and the fifth (Q5) figure[20] of [128]. We chose the queries Q1 and Q2 as those are the simplest queries to produce

[20] In this thesis, we provide updated results for the figures: We used the SPARQL query results for Q1 to generate Figure 4.8, for Q2 to generate Figure 4.9, and for Q5 to generate Figure 4.12.

figures of [128]. We add Q5 to the list, as Q5 is the first query to involve a comparison between snapshots, which considerably rises the number of triple patterns in the query. To estimate the complexity of the SPARQL queries see Table 4.6.

To showcase the applicability of our approach, we present loading and querying times for the first five snapshots and the three queries. We used three different state-of-the-art SPARQL engines with fundamentally different architectures. We do not want to publicly name the two commercially available engines, hence we use abbreviations. We call them Engine V, Engine B, and Linked Data-Fu [198].

From the five snapshots, we yielded about 10 M triples overall with information about the physical level (including the hashes on the graphs) and 481 M triples overall with the reified data (derived from about 80 M triples raw data), the hashes, and data about pay-level domains required for more analyses from [128]. As the snapshots are processed individually and because we exploit in the model the fact that many sources do not change so much over time, there is a high number of duplicate triples between the data from different snapshots. Therefore, the data to be processed by the SPARQL query engines is considerably lower than the reported triple numbers.

We ran the experiments on a Debian 8 (Jessie) 64 bit Linux system with 4 cores of an AMD Opteron 62xx with 2 GHz and 48 GB of RAM. We report the times in Table 4.7. We report the loading times for the physical data (including the hashes on the graphs), and the full data. Moreover, we briefly report on the lessons learned while using different SPARQL query engines to evaluate our approach:

Engine V (Version 7.20.3217) is a SPARQL engine based on a relational database. This engine was not able to load the full data due to the skews in the data introduced by the heavy use of RDF lists for the reified triples. The query performance of the engine was the best in our initial experiments, when considering the evaluation of the queries after indexing.

Engine B (Version 2.1.4) is a SPARQL engine based on specialised indices. This engine managed to load the entire data, but query performance is low when using the automatic query optimiser for queries where different snapshots are compared, for instance it took about a week to get results for Q5. With the feature of the engine to manually optimise the query plan, we were able to significantly improve the performance, for Q5 down to 73 s. In contrast to the other engines we considered, this engine allows to introspect the query evaluation, which helped optimising the query plans.

Linked Data-Fu (Version 0.9.12)[21] [198] contains a SPARQL engine that executes queries on the fly in main memory by streaming RDF through a query plan. Hence, Linked Data-Fu does not employ indices and the availability of main memory limits the size of data to be processed. As Linked Data-Fu does not support aggregates, we had to remove them from the queries and implemented them using GNU AWK[22] scripts.

[21] `http://linked-data-fu.github.io/`, visited on 2018-06-10.
[22] `http://www.gnu.org/software/gawk/`, visited on 2018-06-10.

Table 4.7.: Times for loading and querying data.

	Load		Query			Load+Query physical		
	physical	full	Q1	Q2	Q5	Q1	Q2	Q5
Engine V	**181 s**	failed	**4 s**	**18 s**	**14 s**	185 s	199 s	195 s
Engine B	289 s	**6 h**	45 s	80 s	73 s*	334 s	369 s	362 s
Linked Data-Fu + AWK	n/a	failed‡	97 s	112 s	178 s	**97 s**	**112 s**	**178 s**

* Using manual query optimisation. Otherwise, the execution time was about 1 week.

‡ The engine ran out of main memory when considering the full data set.

For the query evaluation, we piped the results from Linked Data-Fu through AWK 4.1.1 to get the aggregated results. When considering both the loading time and the querying time, Linked Data-Fu proved to be superior to the other two engines we considered.

We report the run times in Table 4.7.

4.8.7. Discussion

While the expressivity of RDF and SPARQL proved to be sufficient for our purposes, the execution time of the queries was not satisfactory for all queries.

We observed that our particular workload poses challenges to indexes and query optimisers. One query engine was not able to load the full data, a different query engine only returned results for Q5 in reasonable time if we interfered with the query optimisation. To work entirely without indexes is fastest if we only need query results once. As we have to do multiple queries, we use the index-based approach employed by the two commercial engines. Although one query engine dominates in terms of run time, the updates required by the semantics of unavailable sources only reliably run on the other engine. Hence, we use both and share intermediary results between the two.

As our goal is to perform the queries on all snapshots, 299 at the time of writing, we have to considerably optimise, given that our initial experiments spanned only five snapshots. For instance, to reduce the number of SPARQL features known to be computationally expensive such as property paths and joins, we added triples with predicate `:hasLastResponse`, see Figure 4.6. Moreover, for the queries of Figure 5 seqq. of [128], we need to compare snapshots. To re-use intermediate results, we could add triples with predicate `:changed`, see Figure 4.6.

More optimisations concern the the representation of the RDF data: For instance, we could reduce the overall data by omitting triples that are depicted dotted in Figure 4.6 because they are the same for all requests or triples. Second, we could use less verbose modelling for the triples that are depicted dashed in the figure: We use the RDF list to describe the HTTP body because it is terminated. The order of the statements does not matter. As we want to use SPARQL for querying, where closed-world assumption (CWA) [181] is made, we can reduce the number of triples by introducing a blank node

Table 4.8.: The Top-5 PLDs with URIs that always returned HTTP status `200 OK`, but never returned data.

PLD	No. URIs
`semantictweet.com`	2 681
`legislation.gov.uk`	202
`dbpedia.org`	159
`oszk.hu`	82
`bio2rdf.org`	81

or URI for the HTTP response body, and use triples with a predicate like `rdfs:member` to connect the body to the statements in the RDF graph of the response body.

For the reasons of unsatisfactory scalability of the SPARQL engines, we report the figures on the dynamics of Linked Data in this thesis in the following fashion: For the least complex queries, i. e. Q1, Q2, and high-level statistics, we report results for 299 snapshots, i. e. 6 years of observation. For the queries with raised complexity, i. e. Q5, Q6, and T2, we report results for 129 snapshots, i. e. $2\frac{1}{2}$ years of observation. For the queries with property paths or queries that involve the logical level, we report the results from 29 snapshots, i. e. $\frac{1}{2}$ years of observation.

4.9. Results on the Physical Level

In this section, for the documents retrieved from the fixed set of seed URIs, we first look at the availability of documents over time, the estimated life-span and death-rate of these documents, their rates of change, and temporal coherence.

4.9.1. Availability/Occurrence

Of the 95 567 URIs, which could be dereferenced successfully in at least one of the 299 snapshots, only 92 248 URIs provided a non-empty HTTP body. We investigated more closely on 129 snapshots for patterns. We report the top-5 providers of URIs that always returned a response with status code `200 OK`, but never returned data. Such URIs hint at misconfigured servers, as there exists the specific status code `204 No Content` for the case of a successful and deliberately empty response. For instance, in the case of `dbpedia.org`, we can look up arbitrary URIs starting in `http://dbpedia.org/data/` and get a `200 OK` response. For such URIs that return an empty RDF Graph, an error status code, e. g. `404 Not Found`, would be appropriate instead. Figure 4.8 shows the distribution of the availability of seed URIs, counting for how many snapshots they eventually responded with `200 OK`, measuring their stability over the 299 weeks. We see that 4 % were available for all 299 weeks of the monitoring period. 80 % of documents were available for 290 weeks or more and the mean availability for documents was 238 snapshots (79.6 % availability). Those figures are in line with the figures we reported

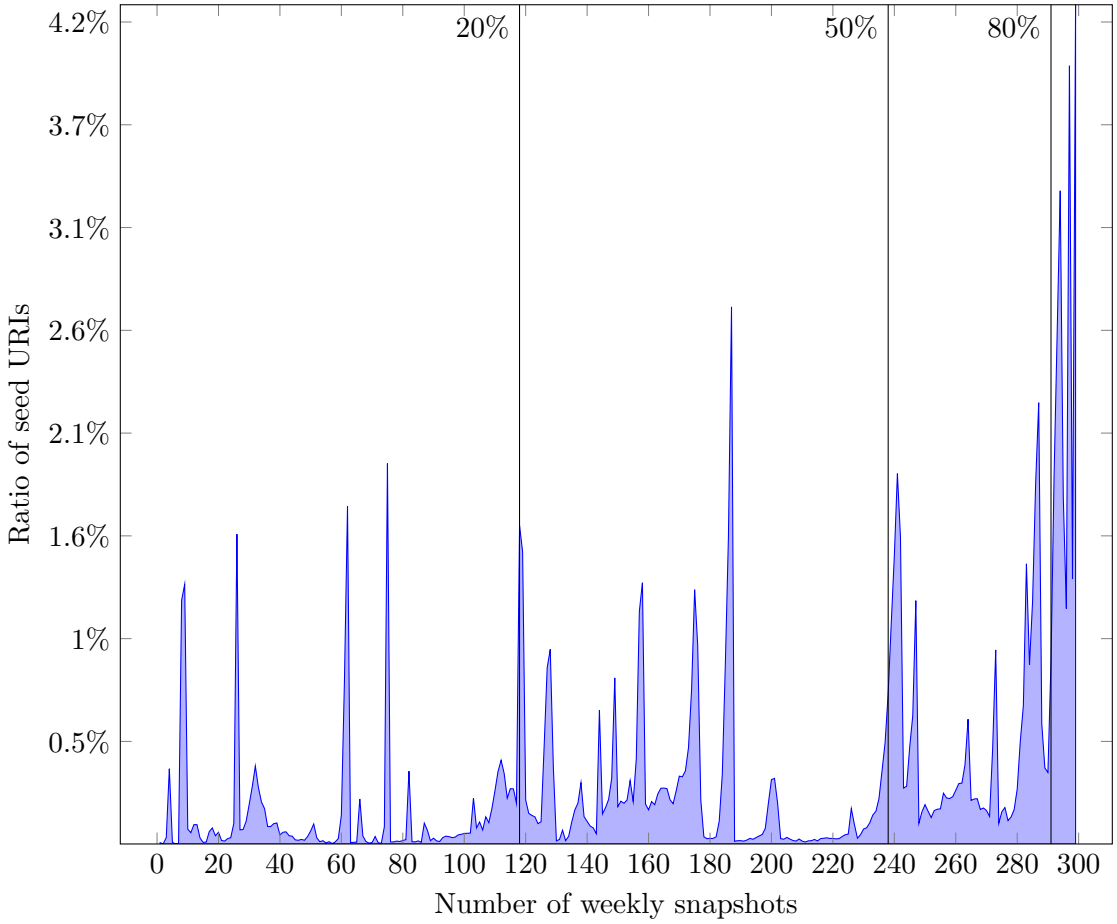


Figure 4.8.: Appearances of documents.

in [128] analysing 29 snapshots.

With respect to this "one-in-five" unavailability of documents, Figure 4.9 provides a breakdown of the HTTP response codes and errors encountered while accessing URIs in the kernel (after following redirects). Response codes in `2xx` are most common: all of these were `200 OK` suggesting that content was returned. The remaining responses indicate errors, where we see increasing instability over the monitoring time-frame. While we observed in [128] that most errors were `5xx` server error codes, those errors are now outnumbered by `4xx` codes, 87.0 % of which were specifically `404 Not Found`, indicating that there is no longer any document at that location. In the next section, we investigate these "dead documents".

Discussion: A one-in-five unavailability rate suggests that an agent traversing Linked Data documents can, on a single pass, expect to miss about 20 % of potential content. This unavailability is not unique to Linked Data: for example, looking at 151 M HTML pages in 2003, Fetterly et al. [72] managed to download only 49.2 % of pages eleven times

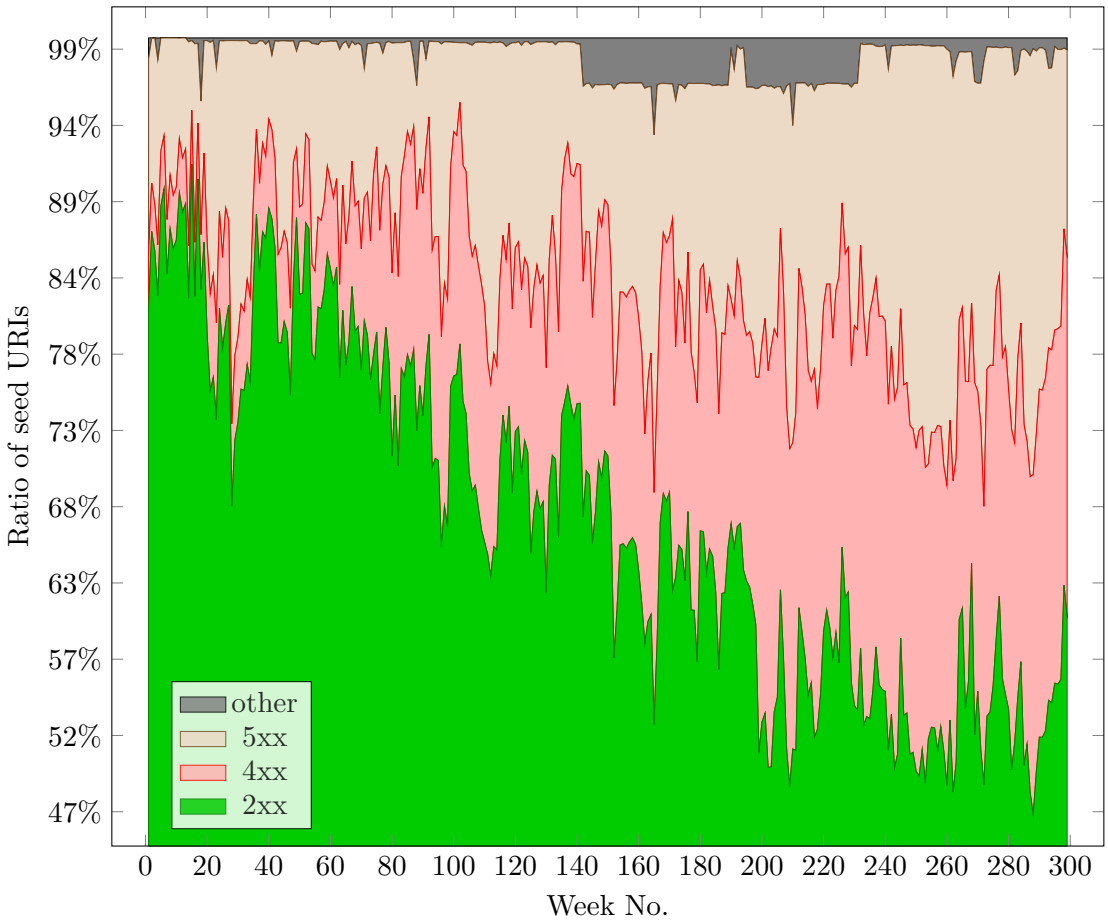


Figure 4.9.: Response distributions. Note that the y-axis does not start at 0 for the legibility of the results.

in eleven weeks; in fact, our results are much more stable by comparison (cf. Figure 4 in [72], and Figure 4.9). One may then ask how often unavailability is temporary, rather than permanent. We next investigate this unavailability.

4.9.2. Death Rate

Given estimates about their stability, we now estimate the loss of documents in the kernel over time by identifying *dead documents*: documents that (are likely to) have gone permanently offline. First, we look at the *last-heartbeat* of documents: the last weekly snapshot in which the document appeared such that, e.g., if a document was last seen in week 2, this document is unlikely to ever return. Figure 4.10 shows the evolving last heart-beats of kernel documents where, e.g., we see that 95 % of documents have appeared at least once since the 14^{th} snapshot (2012-08-05). The further left the life-span, the longer the document is offline and the less likely that it will return. Thus the sharp downward trend observable for the last three snapshots could be due to temporary issues.

Taking another perspective, we also estimate the death-rate of documents by looking specifically at `404` errors that often indicate a permanent error (vs. `5xx` codes that may indicate, e. g. temporary unavailability or server bugs). We found that 98.3 % of URIs that return a `404` error never return content again in our monitoring frame, and 99.7 % of URIs that return `404` errors in two successive snapshots never return. Based on returning a sequence of `404` codes up to the most recent snapshot, Figure 4.11 shows the rate at which documents die in a manner comparable with the analogous "last heart-beat" measures: the `404` death-rate likely underestimates the amount of dead documents (since it does not cover all possible causes), whereas the last heart-beat measure overestimates the amount of dead documents. Combining both perspectives, 5 % of the documents have returned a trailing sequence of five or more `404`s or have been offline for more than 14 weeks, strongly indicating death.

Discussion: The one-in-twenty death-rate of Linked Data documents over six-months is pertinent for link-maintenance (detecting and avoiding dead-links) and for cache maintenance. The death-rate of 5 % over six months can be compared favourably with death-rates of 20.5 % observed by Koehler [142] in 1999 and 48 % observed by Ntoulas et al. [163] in 2004 for HTML documents. We conjecture that since (cool [186]) URIs also serve as names in Linked Data, such documents often have more stable URLs than, e. g. HTML URLs that often contain query strings.

4.9.3. Change Ratio

Next, we compare the RDF content of the documents on a week-to-week basis. For each document, we compare 28 sequential version pairs. If a document was not available for a given week, we make the comparison with the most recent available version of the document. We wish to compare RDF content and not document syntax: thus, our comparison is complicated by the presence of existential blank nodes. In theory, our crawler uses a deterministic mechanism for labelling blank-nodes such that, for a given Web document, the labels of blank nodes will be consistent if blank nodes are consistently

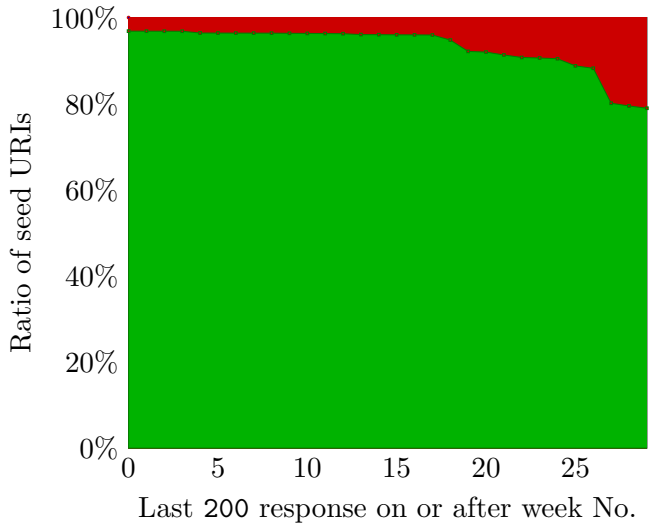


Figure 4.10.: Last heartbeat of documents.

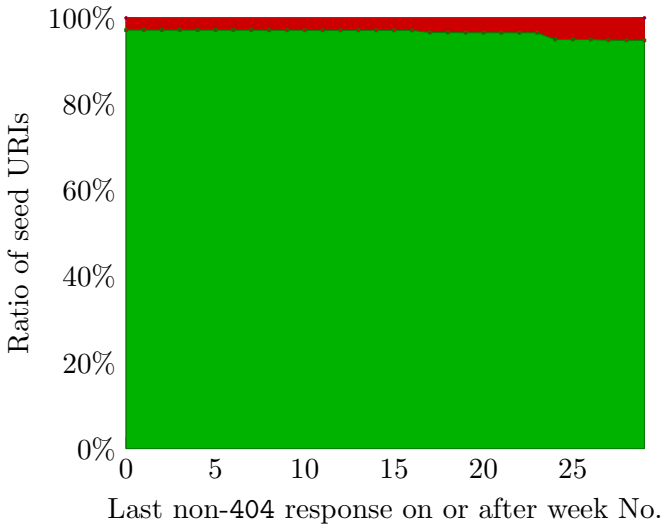


Figure 4.11.: Documents reported dead.

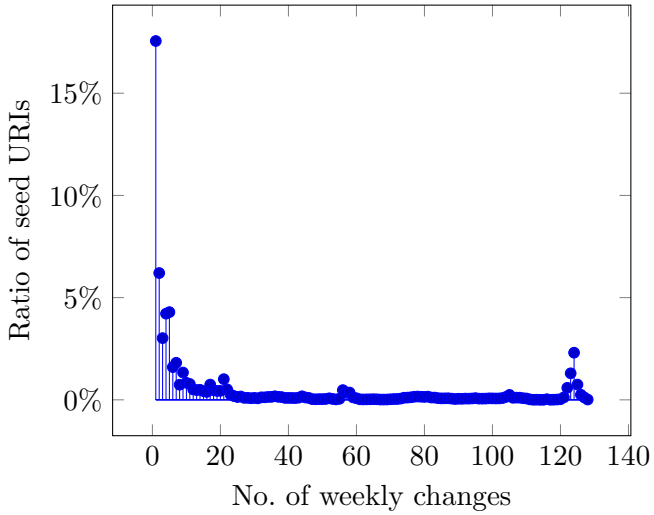


Figure 4.12.: Document change distribution.

labelled in the source document and/or the order of implicit blank nodes remains the same. However, some documents in our collection provide fresh, explicit blank node labels upon every access[23]. With the algorithm of [115] to skolemise blank nodes not yet available when we performed the analyses in this section, we applied an approximation whereby we rewrite all blank nodes to a single, global, fresh constant (i. e. we considering all blank nodes as equal). This allowed us to detect changes in documents, including additions and deletions of statements, irrespective of blank node labels. We compared this approximation to an isomorphism check for RDF Graph equivalence and found that it corresponded in all pair-wise comparisons of document versions for both positive and negative cases.

The distribution of changes for the kernel documents across the 129 snapshots is plotted in Figure 4.12, where we see the ratio of documents with 0–128 changes across the snapshot pairs. At $x = 0$, we see that about 17 % of documents did not change over the 129 weeks (down from 62.2 % when we investigated only 29 weeks). Yet, when looking at a larger number of changes, the results of 29 weeks remain valid: Most URIs fall into the slightly dynamic interval (URIs that change in up until 15 % of the snapshots), followed by the highly dynamic interval (URIs that change in at least 85 % of the snapshots). The rest of the URIs fall into the large remaining middle interval.

Next, we are interested to characterise changes of documents within the same pay-level-domain. In Figure 4.13, we plot domains along two dimensions of change: the x-axis represents the ratio of documents on that domain that exhibited at least one change in the monitoring period, the y-axis represents the mean number of changes for the documents on that domain (including only those that changed at least once), and the size of the tick indicates the number of sampled documents for the domain. We also

[23] See e. g. `http://dbtune.org/artists/last-fm/Baracudas.rdf`; retr. 2013/02/12.

Table 4.9.: Top-10 static PLDs by number of documents.

No.	PLD	No. Documents
1	tfri.gov.tw	2 671
2	culture.gouv.fr	1 499
3	geovocab.org	479
4	beeldengeluid.nl	249
5	kasabi.com	170
6	dataincubator.org	92
7	iandavis.com	88
8	lcsubjects.org	35
9	nytimes.com	33
10	advogato.org	33

annotate some examples of notable domains. 176 domains sit at the origin of Figure 4.13 indicating no changes in any of their 5 656 documents. We investigate those domains in Table 4.9 and see that 5 domains represent 87 % of the documents, and the rest of the documents is on low-volume domains. Further inspection of the domains showed that the domains at the origin are often personal homepages. For the bigger picture and since the majority of domains tend to cluster towards three of the four corners, we consider the following classification of domains:

Steady domains contain a low ratio of documents that change, and these documents change infrequently. Per Figure 4.13, 214 domains fell into the Static quadrant (**32.82 %**), including w3.org, data.gov.uk, and l3s.de.

Dual domains contain a low ratio of documents that change, but these documents change frequently. Per Figure 4.13, only 3 domains fell into the Dual quadrant (**0.46 %**), including indiana.edu and linkedmdb.org.

Bulk domains contain a high ratio of documents that change, but these documents change infrequently. Per Figure 4.13, 306 domains fell into the Bulk quadrant (**46.93 %**), including dbpedia.org, freebase.com, and bio2rdf.org.

Active domains contains a high ratio of documents that change, and these documents change frequently. Per Figure 4.13, 114 domains fell into the Active quadrant (**17.48 %**), including dbtropes.org, loc.gov, and linkeddata.es.

Based on meta-data from the LOD cloud and the DataHub[24], in Table 4.10, we show the breakdown of domains in the categories outlined above for (i) dataset topic, and (ii) whether the data is exported directly by the producer or by a third party. We could not locate topic or producer information for many (non-LOD) domains with few documents

[24] http://lod-cloud.net; http://datahub.io/, visited on 2013-03-08.

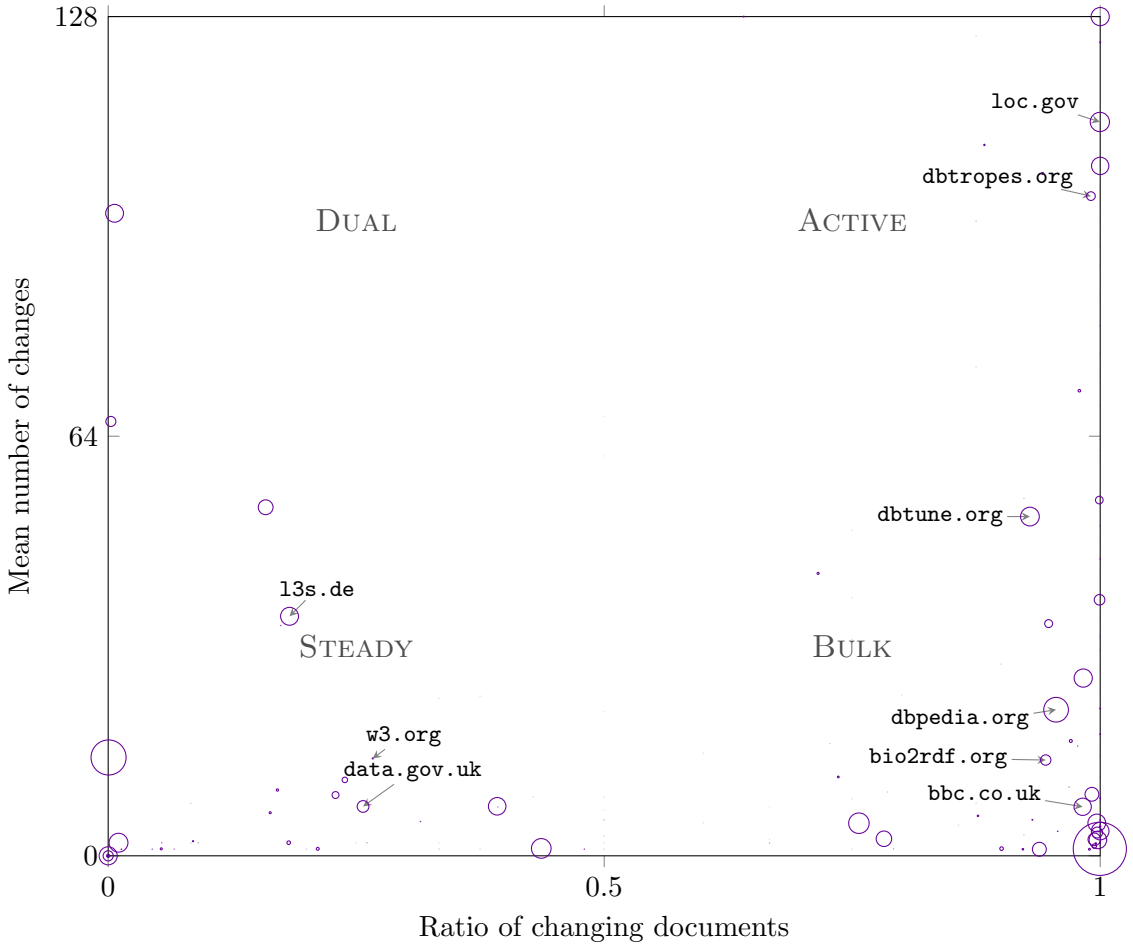


Figure 4.13.: Clustering of domain changes.

Table 4.10.: Dynamicity of Linked Data domains per topic and per party involved

Category	Doc No.	Dom No.	Steady		Dual		Bulk		Active	
			No.	%	No.	%	No.	%	No.	%
cross-domain	29 873	40	17	42.50	1	2.50	13	32.50	9	22.50
geographic	4 840	10	4	40.00	1	10.00	3	30.00	2	20.00
government	5 507	15	7	46.67	0	0.00	5	33.33	3	20.00
life-sciences	8 214	6	3	50.00	0	50.00	3	50.00	0	0.00
media	8 411	28	6	21.42	1	3.57	14	50.00	7	25.00
publications	22 203	36	10	27.78	0	0.00	21	58.33	3	8.33
user-generated	8 523	39	11	28.20	0	0.00	15	38.46	13	33.33
unknown	8 166	478	156	32.64	0	0.00	232	48.54	77	16.11
first-party	26 235	94	28	29.79	1	1.06	46	48.93	17	18.09
third-party	39 531	61	29	47.54	1	1.64	22	36.07	9	14.75
both	22 501	35	9	25.71	1	2.86	15	42.85	10	28.57
unknown	7 470	462	148	32.03	0	0.00	223	48.27	78	16.88
total	95 737	652	214	32.82	3	0.46	306	46.93	114	17.48

Table 4.11.: Static PLDs by topic.

Topic	No. PLDs
cross-domain	9
geographic	2
government	4
life-sciences	0
media	6
publications	4
user-generated	9
unknown	142

(cf. Table 4.10). Since domains may host multiple datasets, if we found multiple topics or production types associated to a single domain, we categorised it as cross-domain or *both*, respectively. In general, we see few high-level patterns in dynamicity for different topics or methods of production. Perhaps most notably, third-party exporters tend to be more active than first-party producers (presumably due to "live exporters"). Also, user-generated domains tended to be more active (though the number of such domains was low).

Discussion: We find that 6.0 % of documents did not change in the 129 weeks and thus are obvious candidates for long-term caching. This finding is in contrast to shorter-term studies, such as our analysis of half a year [128], where we found considerably larger numbers for static URIs (62.2 %). We can conclude that in the long term, a change has to be expected, in mid-term cycles however, it is safe to assume that a majority of the sources is static. Similarly, shorter studies of the HTML web found 56 % of HTML pages to be static, as reported by Brewington and Cybenko [29] in 2000, 65.5 % reported by Fetterly et al. [72] in 2003 and 50 % reported by Ntoulas et al. [163] in 2004. Such works also confirm that past dynamicity can be used to predict future dynamicity in the mid-term. Our work also clusters changes per domain, helping to design synchronisation strategies, where, e. g. a change detected for a Bulk site such as `dbpedia.org` suggests that all documents from that domain should be refreshed. Similarly, Ntoulas et al. [163] showed that change predictions made for individual sites can often (but not always) be accurate. Last, we note that while static domains appear in all of the topics (see Table 4.11), no domain is entirely static (cf. Figure 4.13), and hence we can confirm hypothesis H1.

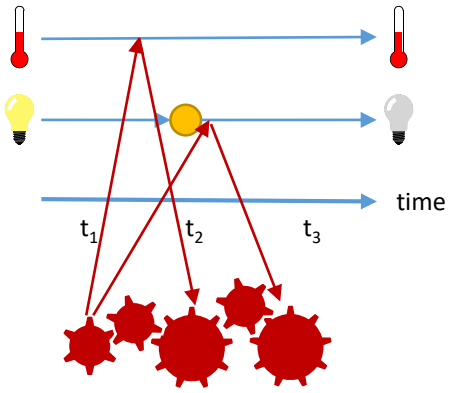


Figure 4.14.: Illustration of the problem of coherent snapshots. A client (gear wheels) determines a snapshot of distributed components (lamp and thermometer). Images from Pixabay and Wikimedia Commons.

4.9.4. Temporal Coherence

Next, we look at the temporal coherence of the snapshots. When working in a distributed environment, it can be challenging to obtain consistent snapshots of the environment, given network and processing delays. In our example, imagine the scenario of Figure 4.14, in which the user agent fails to produce a consistent snapshot. The problem of determining consistent snapshots is well-studied in distributed systems, where e. g. Chandy and Lamport proposed an algorithm to determine a consistent snapshot of a distributed system [39]. The assumption of this body of work is that the components in the distributed system can run some algorithm on behalf of the user agent that wants to determine a consistent snapshot. The algorithm run on the components allows for determining the snapshot. On the web, however, we only assume HTTP-GET access to system components and hence we cannot assume that we can run algorithms on arbitrary system components. On the HTML web, the problem has also been subject to scholarly investigation, e. g. by Spaniol et al., who coined the term *temporal coherence* for this problem [194].

Example 8 (Temporal Coherence)**.** *Imagine a user agent that operates in our example of Internet of Things devices. To reason about his actions, the user agent determines a snapshot, see Figure 4.14. The user agent (gear wheels) starts at t_1 to determine the snapshot of the distributed components (lamp and thermometer). At t_2, somebody turns on the lamp. The different components are subject to different network delay. The component with the thermometer responds quickly. The component with the lamp has a worse connectivity, and hence answers later. As the response from the component with the lamp has been sent after t_2, the point in time of the state change of the lamp, the combined responses of the lamp is incoherent (t_3).*

To investigate the temporal coherence at a broader scale than our example, we look at the temporal coherence of the snapshots of the Dynamic Linked Data Observatory. In contrast to our example in Figure 4.14, we consider not 2 resources but the seed list

of 95 737 resources, from which we want to obtain a snapshot. In contrast to previous work [194], where the goal was to order a list of URIs for downloading such that temporal coherence is guaranteed, we assume that the number of URIs is sufficiently large such that temporal incoherence will occur. Hence, we want to quantify the error introduced by temporal incoherence.

We base our investigations on the metric *age* defined by Cho and Garcia-Molina to quantify how out-dated a copy of an element e_i (i. e. a page on the web) and a collection of copies of elements is [41]. For temporal coherence, all copies in the collection should have the same age.

Definition 8 (Age)**.** *The age A of a copy of a page is the time that passed since the first update after the copy was made, i. e.:*

$$A(e_i, t) := \begin{cases} 0 & \text{if } e_i \text{ is up-to-date at time } t \\ t - \text{modification time of } e_i & \text{otherwise} \end{cases}$$

The age of a collection of copies is the average of the age of all copies.

To study the age of copies, we need an estimate how the original changes. In line with previous work [28, 42, 44], which investigated the changes of web pages, we use a Poisson process to model the changes of Linked Data documents. The probability density function of a Poisson process that models events can be given as

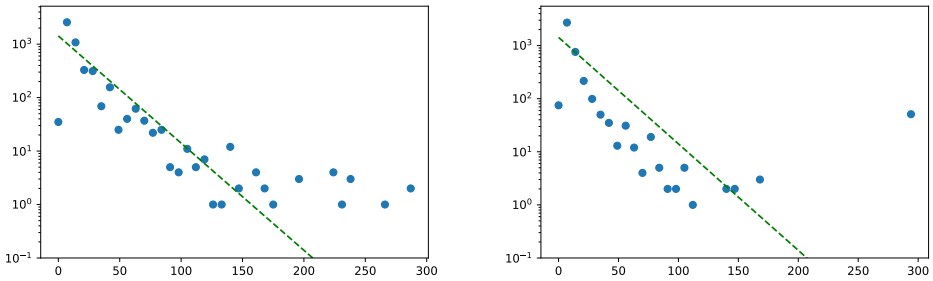
$$f(t) := \begin{cases} \lambda e^{-\lambda t} & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

where λ is the average rate at which events occur.

Verification of the Poisson Process In this section, we look at whether our assumption of a Poisson process is adequate. Under a Poisson process, the time between two events is exponentially distributed. Hence, we should expect many changes in shorter intervals, and only few changes in longer intervals. In Figure 4.15, we plot for two given number of changes in Linked Data documents the occurrence of different intervals between changes of Linked Data documents[25]. On the abscissa, we give the duration of the interval between two changes of a Linked Data document. On the ordinate, we give the frequency of a duration as it occurred observing the URIs in the seed list. As the ordinates are logarithmic, exponential functions draw straight lines in the plots. We consider the assumption of a Poisson process to be adequate, as the frequencies of durations between changes are close to an exponential line.

Estimation of λ Facing a similar setting as ours with incomplete change histories (we do not know whether a source that change between two observations changed once or multiple times between the observations) and irregular access intervals (the odd technical problem

[25]The plots of other numbers of changes look similar, given that sufficient data is available.



(a) $n = 36$ (b) $n = 46$

Figure 4.15.: Frequency over the duration between two changes for sources with n changes, and an exponential function fitted using the method of least squares. Note the logarithmic scale on the y-axis.

during the six years of observation sometimes caused delays creating the downloads), Cho and Garcia-Molina developed an estimator that is in this setting superior to the naïve estimator for λ, which divides the number of changes by the overall timespan of the observation [44]. Cho and Garcia-Molina's estimator estimates λ indirectly, by estimating $r = \lambda/f$, the ratio between λ and the access frequency f. We use the this estimator by Cho and Garcia-Molina and estimate r using:

$$\hat{r} = -\log\left(\frac{\bar{X} + 0.5}{n + 0.5}\right)$$

where $\hat{r}$ is the estimated r, n is the number of accesses, X is the number of observed changes, and $\bar{X} := n - X$ is the number of accesses without an observed change. We then can compute the estimated λ, $\hat{\lambda}$, using:

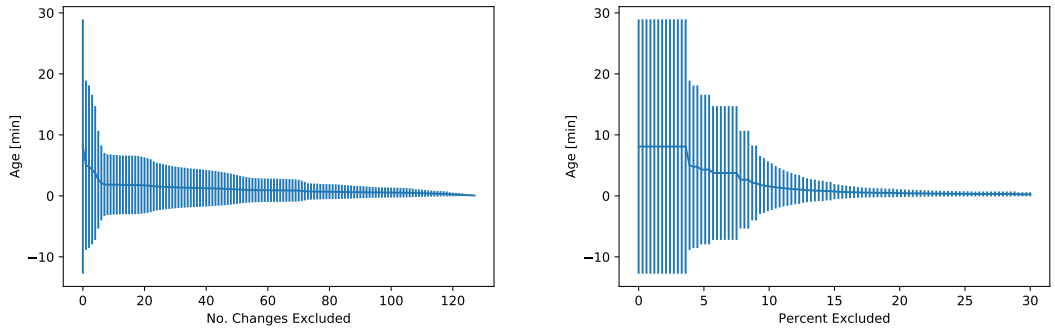
$$\hat{\lambda} = f \cdot \hat{r}$$

Expected Value of Age From λ, we can compute the expected value for the age for a source e_i as follows [41]. With $s \in (0, I)$ the time point of modification in the interval until I, the time of the next access, the age at time $t \in (s, I)$ is $(t - s)$. Hence, the expected age of element e_i at time $t \in (0, I)$ is:

$$E[A(e_i, t)] = \int_0^t (t - s)(\lambda e^{-\lambda s})ds = t\left(1 - \frac{1 - e^{-\lambda t}}{\lambda t}\right)$$

Results and Discussion We calculated λ and $E[A(e_i, t)]$ for all 95 737 seed URIs over 129 snapshots for the case that the downloads last 10 h, cf. Figure 4.5. Then, we derived the expected value for the age of all individual sources, and the average of the set of seed URIs and find:

$$E[A(\text{snapshot}, 10\text{h})] = 8{:}05\text{min} \pm 20{:}51\text{min}$$



(a) Excluding the highest change frequencies achievable

(b) Excluding the top-x% dynamic URIs

Figure 4.16.: Mean and standard deviation of the age of URIs in the snapshot.

That is, an application that uses a snapshot from the Dynamic Linked Data Observatory has to take into account that the data from different sources stems from time points that are at average about 8 min apart. As we will see analysing the RDF terms, the changes in the most dynamic sources are timestamps of the generation date (cf. Table 4.12), i. e. rather changes to the meta data than to the actual data. Hence, we look at two options to exclude URIs whose changes we consider meta data changes. First, we exclude the URIs that have the highest change frequencies possible, i. e. we increase n in $N-n$, where N is the total number of snapshots, see Figure 4.16a. Second, we exlude the top-x % dynamic URIs, see Figure 4.16b. We observe that already by excluding the frequency that equals the number of snapshots, we can achieve a considerable drop in mean and standard deviation, see Figure 4.16a. Looking at Figure 4.16b, we see that by excluding the highest frequency, we disregard about 4 % of all URIs. If we account for this by only considering the URIs which changed in 96 % of the snapshots or less, we find:

$$E[A(96\,\%\text{-snapshot}, 10\text{h})] = 2{:}07\text{min} \pm 6{:}08\text{min}$$

Discussion We provided a statistical analysis of the Dynamic Linked Data Observatory for temporal coherence of the snapshots of the observatory. Under the assumptions of a Poisson process and a download time of 10 hours, we see that during the 10 hours, the dynamics of the sources introduce an age of the snapshot of 8:05min (2:07min if we exclude suspiciously dynamic sources). While we hence cannot guarantee temporal coherence, we argue that an incoherence of 1.3 % (0.3 %) of the overall download time introduces an error hopefully negligible for the use-case of the download.

A mitigation strategy to avoid incoherence is to optimise the order of the URIs during the download for coherence, as described in [194]. In contrast, the download of URIs using the crawler used for the Dynamic Linked Data Observatory does not follow a particular order, except that the crawling process is optimised towards polite crawling. Other strategies include to reduce the download time by caching the least-frequently

changing sources, or to investigate whether the list of URIs can be reduced by focussing on the specific use-case, e. g. using techniques from focussed crawling [37, 60].

4.10. Results on the Logical Level

We see that Linked Data documents change with varying degrees of breadth and frequency on different domains, and that documents on some domains, such as `dbtropes.org`, change widely and often. We now look at what kinds of changes are occurring on an RDF-level within these documents by considering the first 29 snapshots from the Dynamic Linked Data Observatory.

4.10.1. Triple-Level Changes

We first look at the types of changes for documents. We found that 27.6% of documents only ever updated values for terms (one per triple) in the RDF Graph they contain across the 29 weeks, keeping the number of triples static: such changes would include, e.g., updating a literal value like as an access-date entry. A further 24.0% of documents only added triples across the 29 weeks, representing monotonic additions. Changes for other documents involved a mix of additions, single-term updates and deletions across the different weeks.

In Figure 4.17, we plot the ratio of documents for which we found at least one triple addition over the 29 weeks against the ratio of documents for which we encountered some deletion over the 29 weeks, looking for high-level patterns. For the purposes of this plot, we consider a term update as an addition and a deletion of a triple. We see that most of the domains fall along a STABLE line where an equal number of documents involve some additions and some deletions: again, many of these documents correspond to the 27.6 % that only update individual values (an add and a delete). Close to the $(0, 1)$ point, we see two large "monotonic" domains (`europa.eu` and `gesis.org`) in the GROWING half-space that almost always only ever *add* triples to their documents. The one notable domain in the SHRINKING half-space was `bio2rdf.org`, for which 52 % of documents had additions and 85 % had deletions.

Discussion: For Triple Stores, additions are often cheaper than deletions (especially if inference and truth maintenance are required). Here we see that additions to Linked Data documents are almost always accompanied by deletions, emphasising the importance of efficient revision strategies for warehouses. In relation to the HTML Web, Brewington and Cybenko [29] show that the content of HTML pages tends to grow over time, though their results rather reflect technological trends over a period of many years (1995–1999). The shrinking of the `bio2rdf.org` data set can be accounted to the release 2 of `bio2rdf.org`, where the maintainers consolidated the data set [33] during the timespan of our observation.

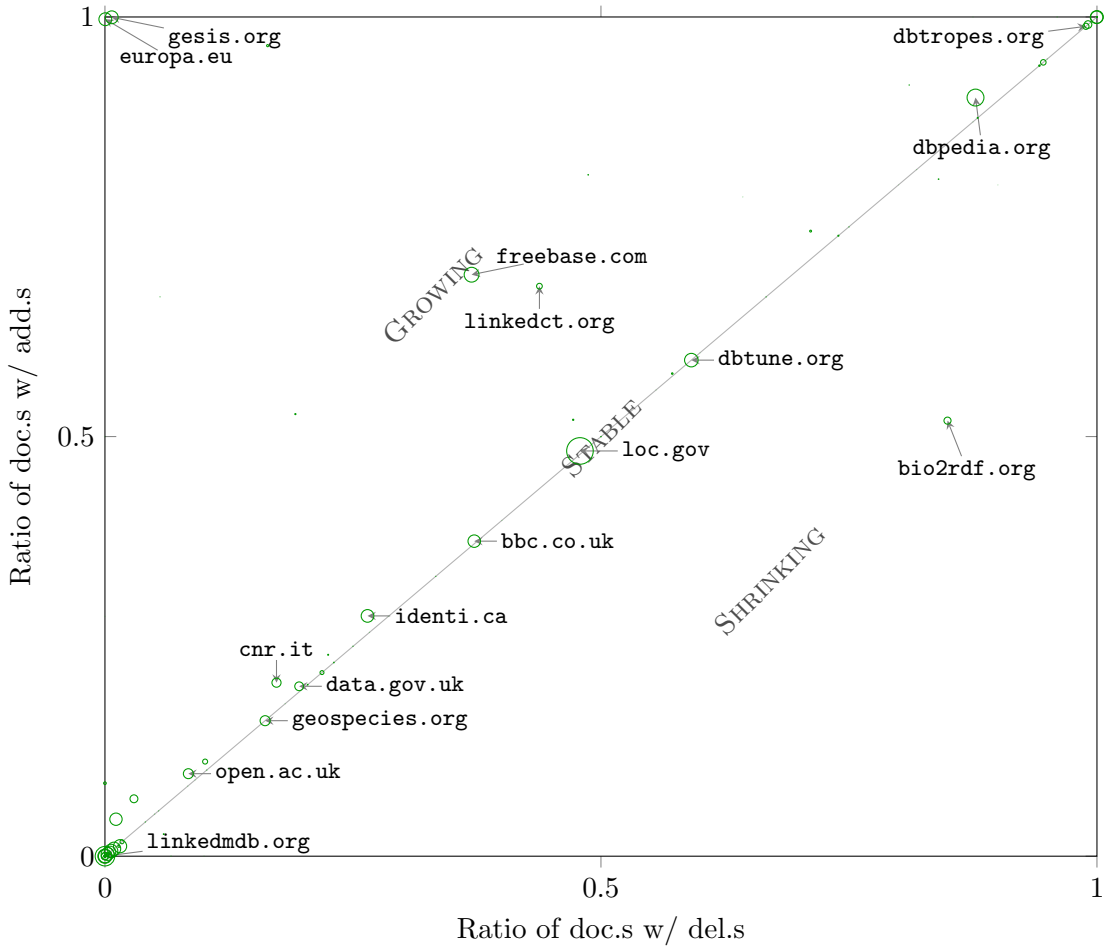


Figure 4.17.: Ratio of documents with additions vs. deletions per domain.

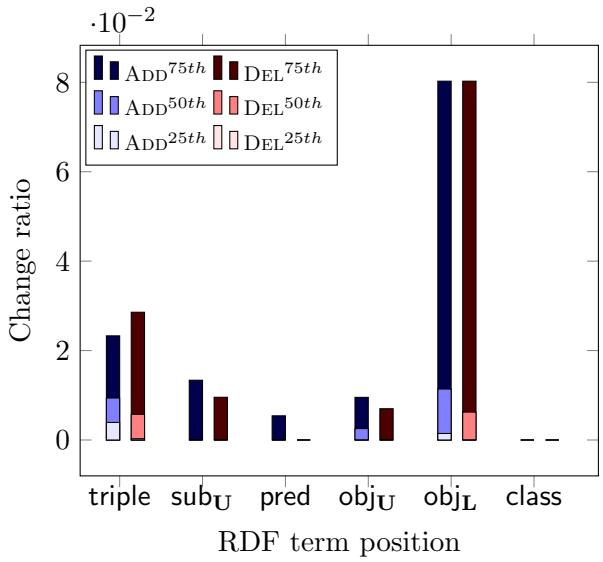


Figure 4.18.: Additions (left) and deletions (right) for different RDF elements.

4.10.2. Term-Level Changes

Next we look at the types of terms changing in the RDF content of the kernel. Figure 4.18 plots the 25^{th}, 50^{th} and 75^{th} percentiles[26] for the addition/deletion of RDF triples and the terms they contain. We only consider documents that changed at least once in the 29 weeks and omit blank node terms due to possible homomorphisms (relying on our approximation for *triples* involving blank nodes). We compare changes to subject URIs, predicates, object URIs, object literals and classes (values for `rdf:type`). The y-axis reflects the ratio of triples or terms that changed versus the total number of unique such elements observed in the documents considered (the y-range is small: $[0, 0.08]$). A ratio of 0.08 for object literal additions thus indicates that, over 29 weeks, the number of unique object literals added to the documents at that percentile was $0.08\times$ the total number of unique object literals appearing in those documents.

We see some clear trends. First, we see that additions and deletions are often of a similar magnitude, reflecting back on previous observations about terms often being directly replaced. Second, the most dynamic position of an RDF triple is the object, with a high churn of object literal values in particular. Conversely, predicates are occasionally added but rarely removed from documents. Analogously, class terms are very rarely added and very rarely removed (barely even seen above the x-axis). These latter two observations suggest that the *schema signature* of documents (set of property/class terms used) is generally static.

Discussion: The types of terms that change offer interesting high-level patterns into the dynamicity of RDF in general. For example, the observation that the set of properties and classes instantiated by a document rarely changes lends empirical strength

[26]Higher percentiles cause too much compression of the results; hence we omit them.

to proposals for schema-level summaries of data, such as proposed by Stuckenschmidt et al. [200]. Conversely, we see that literals are the most dynamic element of RDF. The following section sheds light on why this might be the case.

4.10.3. Dynamic Predicates

Though we have seen that predicates themselves are rarely added or removed, we are interested to see which predicates are indicative of dynamic statements. Table 4.12 presents the ten most dynamic predicates according to the ratio of added (+) and deleted (−) statements involving that predicate, divided by the total number of statements for that predicate across all snapshots; we only include predicates that appear in all snapshots and appear in ≥1 000 statements overall. Where added and deleted ratios correspond closely, this suggests frequent "value updates". The two `dbtont:` predicates are used on the third-party `dbtropes.com` domain to indicate a time-stamp since the relevant data were parsed or fetched from the original source (`tvtropes.org`); the `swivt:`, `prv:` and `linkedct:` predicates also provide time-stamps indicating the last time data were refreshed for documents on various domains. The two `sioc:` predicates are used to track dynamic discussions and posts on the social `gnoss.com` domain. The `media:image` predicate appears for RDFa image meta-data, most of which are embedded in `msn.com` news pages. The `xhtml:bookmark` predicate represents links embedded as RDFa in various dynamic XHTML pages.

Discussion: Identifying dynamic predicates allows Triple Stores to know, in a granular fashion, which parts of an input query relate to static knowledge and which parts to dynamic knowledge (e. g. see the previous proposals by Umbrich et al. on this topic [205]). Per our results, when considering cached content, the ratio of additions indicates the potential to miss answers involving triples with a given predicate, and the ratio of deletions indicates the potential to return stale answers. With respect to the most dynamic predicates, we identify that they are often trivial time-stamps. Comparatively, Fetterly et al. [72] and Ntoulas et al. [163] both discuss how the majority of changes in HTML documents are very minor, involving hit counters, time-stamps, etc.

4.10.4. RDF Link Structure

Finally, we look at the evolving nature of the link structure of documents over time. We first want to see if the overall level of links tends to increase or decrease over time, and are interested to see at what rate fresh links are added to the kernel. We consider any URI in any position of a triple as a potential link from the kernel. Figure 4.19 plots the evolution of the volume of such links over time. We see that the number of links can fluctuate based on the availability of documents (with parallels to e. g. response code distributions for each week illustrated in Figure 4.9). A second key observation is that the ratio of fresh URI links added to the kernel is in general quite small: we consider a URI as fresh if it has not been seen for *any* kernel snapshot before. This clearly indicates that the outward link structure of the kernel remains quite static (aside from instability) over time. In fact, if anything, links are on a decreasing trend as documents die off.

Table 4.12.: Top-10 dynamic predicates.

No.	Predicate	Total	+	−
1	`dbtont:parsed`	35 911	0.94	0.94
2	`sioc:has_discussion`	3 171	0.87	0.99
3	`sioc:content`	107 387	0.87	0.98
4	`dbtont:fetched`	34 894	0.53	0.53
5	`swivt:creationDate`	35 295	0.53	0.53
6	`media:image`	1 377	0.49	0.49
7	`prv:performedAt`	16 706	0.45	0.45
8	`xhtml:bookmark`	17 852	0.45	0.44
9	`linkedct:p.t.u`*	2 652	0.42	0.42
10	`linkedct:p.t.a`*	2 652	0.42	0.42

* `provenance_time_updated` and `provenance_time_added`, respectively.

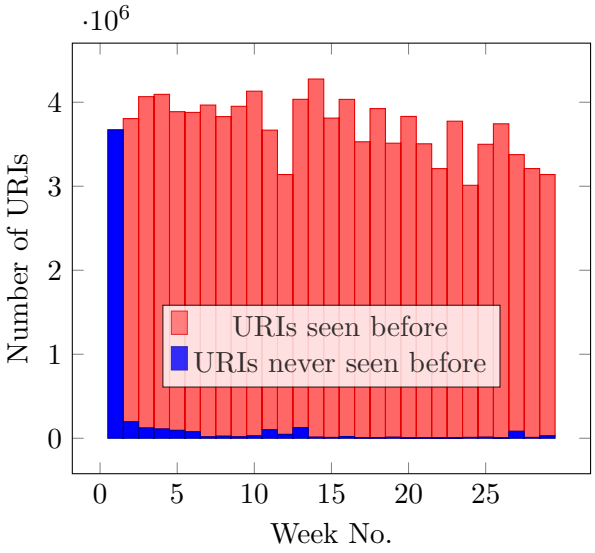


Figure 4.19.: Links extracted from kernels.

That said, after the initial stabilisation over the first month of observations, we do find that a few domains are consistently contributing some fresh links to the kernel (as starts or as ends of links): `sec.gov`, `identi.ca`, `zitgist.com`, `dbtropes.org`, `ontologycentral.com` and `freebase.com` offer a low-volume but constant stream of fresh outward links from week to week. Other domains—including `bbc.co.uk`, `bnf.fr`, `dbpedia.org`, `linkedct.org`, `bio2rdf.org`—tend to introduce new links in batches, corresponding with the update characteristics of domains plotted previously in Figure 4.13. However, such domains are the exception rather than the rule.

Discussion Knowledge about how links change over time is important for any agent that traverses Linked Data documents (in terms of reachability, discoverability, etc.) or analyses link structure (e. g. to compute PageRank). Ntoulas et al. [163] found that hyperlinks in HTML documents tend to be more dynamic than other forms of content, estimating that 25 % of links are new each week (though considering a growing set of documents). In comparison, our results seem much more static. This seems counter-intuitive in that Linked Data itself fundamentally comprises URIs and thus links; however, we already saw that URI terms in RDF documents change slowly (e. g. compared to literals).

On a side note, we also analysed some other parameters of the kernel graph and the extended graph over time, but found them to be relatively static. We found that the document-level graphs invariably formed a single weakly connected component across the snapshots: URIs sampled from the examples for LOD datasets were all (weakly) connected through the expansion of the crawl used in the original sampling. We also checked the diameters of the document-level graph and found that it varied from between 11–12 for both the kernel and, interestingly, for the extended crawls: this suggests that the extended crawls intuitively tend to make the graph "more dense" as opposed to "wider".

4.11. Community Validation

Since we started the Dynamic Linked Data Observatory in 2012, numerous papers have cited our papers. As of 2018-07-15, Google Scholar lists 103 citations for [128] and 55 citations for [134]. Of those papers, the works by the following authors used the data set from the observatory to make own analyses: Gottron and Gottron [87, 86], Dividino et al. [56, 57, 54, 55], Schaible et al. [187], Fernández et al. [70], Nishioka et al. [159, 160, 161], Abdel-Qader et al. [1], and Meroño-Peñuela et al. [154].

Moreover, Fernández et al. built the BEAR benchmark for RDF archives [69] on 58 snapshots from the Dynamic Linked Data Observatory. Hence, papers using BEAR partially build on our works, e. g. works by Cerdeira-Pena et al. [36], Taelman et al. [202], Anderson et al. [4], and Fernández et al. [68].

Given the impact of our work, we conclude that we addressed the challenges presented in Section 4.3 to an extent that is satisfactory for the Semantic Web community.

4.12. Summary

In this chapter, we studied the dynamics of Linked Data on the web using the Dynamic Linked Data Observatory. We made the case for the Dynamic Linked Data Observatory, presented the design of the observatory, described a method to analyse dynamic Linked Data, and provided analyses to shed light on the dynamics of Linked Data on the web.

Using the Dynamic Linked Data Observatory, we target both Linked Data consumers and producers. We derived use-cases (Section 4.1) for knowledge about the dynamics of Linked Data and corresponding research questions (Section 4.2) for both parties to be answered using the observatory. The use-cases and questions span different levels of processing Linked Data: From dynamics on the physical level of accessing Linked Data documents over the network via HTTP, to the logical level of the contents fo the RDF graphs.

To design the Dynamic Linked Data Observatory, we described challenges (Section 4.3) and provided an extensive survey of literature of how to set up such an observatory (Section 4.5). Based on an in-depth statistical analyses of Linked Data on the web, the literature, requirements from our use-cases, and practical considerations, we described the design of the Dynamic Linked Data Observatory (Section 4.6). Using the description of the systematic set-up of the Dynamic Linked Data Observatory, we answer the overall research question of this thesis RQ2 of *how can we construct a corpus to verify and study the dynamics of Linked Data*.

For the concise definition of our analyses, we investigated a method that uses Semantic Technologies (Section 4.8). Specifically, we provide an RDFS model of dynamic Linked Data, and corresponding SPARQL queries to answer the research questions on the level of accessing documents. We applied the method to analysing Linked Data on the physical level and encountered scalability issues that prevented us from also using the method on a logical level.

Last, we provided the results of our analyses (Section 4.9 and 4.10). Based on the monitoring 95 737 URIs for 299 weeks (please find more high-level statistics in Section 4.7), we found that documents were unavailable about 20 % of the time. We then determined that 17 % of documents had no change in that time, where other documents either changed very frequently (5.7 %), or very infrequently (46.3 %), with few documents in between. Based only on 29 snapshots, we can say that of the documents that did change, many updated individual RDF terms in the document (27.4 %) or only ever added triples (23.1 %). We found that domains tended to be either very static (44.5 %), have a high ratio of documents that change infrequently (28.2 %), or have a high ratio of documents that change frequently (25 %); most domains contain a balance of documents with additions and deletions. With respect to the types of changes occurring on an RDF level, we found that object literals were the most liable to change (0.01× ratio for median/50^{th} percentile; 0.08× for 75^{th} percentile), whereas the schema signature of documents –involving predicates and values for `rdf:type`– changed very infrequently. We identified predicates involved in the highest ratio of new/removed triples and found that they often relate to time-stamps. Finally, we showed that the rate of fresh links being added to the documents is low, and varies between 4 960 to 126 944 per week depending

on whether domains with bulk update behaviour were updated.

In terms of connecting these observations back to our original use-cases outlined in Section 4.1, we make the following observations:

Focused Synchronisation: We identified the general rate of change of documents, and found that dynamicity tended to follow certain predictable patterns for PLDs. For example, static and steady domains infrequently require light synchronisation, bulk domains occasionally require heavy synchronisation, dual domains require frequent light synchronisation, active domains require frequent heavy synchronisation (or live querying techniques).

Smart Caching: Reversing the previous use-case, we found that 17 % of documents did not change over the $2\frac{1}{2}$ years and found that 32.5 % of domains were considered static (and thus are candidates for long-term caching). Applications that rely on a schema-level index or schema-level cache of documents can rest assured that the schema-signature of documents tends to be very (though not completely) static. Furthermore, we identified particular predicates whose triples should not be cached due to high rates of updates.

Adaptive Processing: A hybrid architecture could be built along a number of logical data partitions. First, we showed that domains tend to fall into a few clusters, where domains with static, steady and bulk update behaviour could be supported by heavy materialisation approaches, whereas active domains are best supported through decentralised live-querying approaches. Conversely, we also showed, for example, that different schema patterns in the data were indicators of different levels of dynamicity, where partitioning could be done on a per-predicate basis instead, etc.

Temporal Coherence: We investigated the temporal coherence of snapshots and found only a small error introduced by the dynamics of Linked Data on the web. Nevertheless, if temporal coherence must be guaranteed, it should be possible to mitigate the small errors through URI ordering when composing a snapshot.

Link Maintenance: We found instability in documents, though much of this instability was of a temporary nature. However, we found that 5 % of documents had died off during our monitoring period, suggesting an initial estimate for the arrival of deadlinks.

Versioning: We have not tackled the issue of versioning in depth. Some conclusions could be applied incidentally to the area of versioning (e.g. about the frequency of change of different types of RDF terms and the balancing of additions vs. deletions), but further more specialised analyses of the data (by us or the community) would be needed to generate concrete guidelines. Moreover, works e. g. by Dividino et al. [57] and Gottron et al. [87] specifically looked at schema-level changes in Linked Data on the web using the Dynamic Linked Data Observatory, which could also inform versioning strategies.

5. A Model of Computation for Linked Data

Parts of this chapter have been published in:
▷ Tobias Käfer and Andreas Harth. "Rule-based Programming of User Agents for Linked Data". In: *Proceedings of the 11th International Workshop on Linked Data on the Web (LDOW) at the Web Conference (27th WWW)*. 2018.

In this chapter, we go beyond the descriptive (or positive) treatment of dynamic Linked Data of the previous chapters and take a prescriptive (or normative) standpoint, i. e. we investigate a way that allows for describing what behaviour we want to happen in Linked Data. To this end, we develop a model of computation for Linked Data, ASM4LD, that allows us to define the behaviour of user agents that interact with Linked Data sources. We base the model of computation for Linked Data on Abstract State Machines (ASM), a model of computation rooted in mathematical logic. We provide a synthesis of ASMs and Linked Data and show how the combination aligns with the relevant specifications such as the Request/Response communication in HTTP, the Linked Data Platform (LDP) specification for updating resource state, and RDF model theory as formal grounding in logic. Based on the formalisation of changing Linked Data resources, we present the syntax and operational semantics of a small rule-base language to specify user agents that use HTTP to interact with the environment. We show the approach to be Turing-complete. We show the feasibility of the approach in an evaluation involving the specification of automation in a Smart Building scenario, where the presented approach serves as theoretical foundation. Moreover, we present how we used the approach to build a Virtual Reality system, in which sensor data is integrated with data from the Web.

Example 9. *Consider our example with the Internet of Things devices with the lamp and the temperature sensor. How can we define application logic that, e. g., turns on the lamp if the temperature is below* 20 °C *and off otherwise? We consider more complex examples in our evaluation, see Section 5.8.*

5.1. Motivation

Web standards around Linked Data such as HTTP [75], RDF [47], RDFS [30], and OWL [219] facilitate interoperability at a vast scale on the web. For instance, on the level of data transfer, we have access to billions of resources on the web uniformly using the transfer protocol HTTP. On the level of describing data, billions of triples[1] about

[1] `http://lodlaundromat.org/`, visited on 2018-06-10.

resources can be found on the web that have been published uniformly in the data model RDF, enriched with ontological information in RDFS and OWL [85]. On such Linked Data, which is traditionally read-only, developers can build data integration systems based on queries and ontologies, i. e. approaches grounded in mathematical logic, where most of the operations are carried out by query processors and reasoners. However, to integrate such data from read-only sources is only a first step; many scenarios, for example on the Web of Things, require to change data. But while the W3C's LDP (Linked Data Platform) specification [196] combines read-only Linked Data (via the HTTP-GET operation) with read-write capabilities (HTTP-PUT, POST, and DELETE operations) for the interface to writeable resources, most user agents that interact with LDP servers are still programmed in imperative code. We would like to specify the behaviour of user agents in a language grounded in mathematical logic.

A high-level language for the specification of the behaviour of user agents with a formal grounding would allow for expressing executable specifications, which are shorter and cheaper to create than imperative code, and which are easier to validate (e. g. with subject matter experts) and to analyse (e. g. using formal methods). Moreover, the language could also serve as execution substrate for agent specifications (the showcase of the seminal Semantic Web article [22]) created using planning approaches from Artificial Intelligence.

Several approaches exist for specifying user agents on read-only Linked Data. These user agents process queries and have the ability to follow links, which can lead to the discovery of additional data during query processing [24, 105, 98, 206, 106]. An extension to read and write capabilities in addition to the following of links, which is core to web architecture [73, 166], would allow for user agents that are able to not only query data but also to effect change. How to beneficially combine current technologies around Linked Data and REST into a unified formalism of dynamics is an open research problem [111, 217, 171, 100]. Hence, we cover research question RQ3 (how can we specify computation using Read-Write Linked Data and rules) in this chapter by presenting contribution C4, ASM4LD, a model of computation for Linked Data.

5.2. Challenge

An executable formal approach to Linked Data user agents should respect the relevant standards and practices around Linked Data such as HTTP-based data access and manipulation, hypermedia-style link following, the model-theoretic formal foundations of RDF, and allow for semantic reasoning using ontologies. Moreover, the approach should have a formal foundation with a sufficient expressivity to build applications while at the same time be feasible to be applied in practice.

5.3. Contributions

Our contributions are as follows:

- We provide a formal account of the standards around Linked Data in combination with Abstract State Machines (ASMs) [90], a model of computation rooted in mathematical logic. We believe that ASMs are the right starting point: ASMs encode state in first-order logic, which is a superset of the RDF-family of languages; on top, ASMs require only rules to represent state change, and rules have been proposed for RDF data processing [20, 119]. With our synthesis between Linked Data, which provides a way to represent and manipulate resource state, and ASMs, which provides operational semantics based on rules and stepwise execution of these rules, we are able to specify user agent behaviour with the entire user agent state maintained in RDF on web-accessible resources.
- We present a syntax for a rule language (based on Notation3 [20], of which the RDF Turtle language is a specialisation) to define conditions that trigger updates based on HTTP state manipulation operations. We give an operational semantics of the language based on the ASM execution semantics.
- We provide a benchmarking system from the building automation domain and conduct an evaluation of a prototype rule processing system. Moreover, we show the applicability of our approach.

5.4. Example Scenario: Rule-based Control for Building Automation

We sketch a behaviour specification for the functionality of Example 9 in our example scenario with two Internet of Things devices in a room. We give the intuition of the syntax and the semantics of the rule language employed. The formal treatment of the rule language is subject of this chapter.

5.4.1. Intuition of the Syntax of a Condition-Action Rule Language

We express rule programs in a subset of the Notation3 (N3) syntax [20], for an example see Figure 5.1. N3 is a generalisation of the RDF Turtle syntax [180], hence we can express RDF in N3. To form rules, we use N3's graph quoting (curly brackets) and variables (terms starting with question marks). Note that in contrast to RESTdesc [217], which also uses the N3 syntax, we do not interpret the rules as input and output descriptions of HTTP interactions, but as executable specifications.

We distinguish derivation rules and request rules. Rules consist of a rule body, an implication symbol, and a rule head. In both types of rules, the body consists of a set of triple patterns (triples with variables, cf. triple patterns and basic graph patterns in SPARQL [93]). In a derivation rule, the head also consists of a set of triple patterns (cf. (2) in Figure 5.1, which defines the transitivity of `sosa:hosts`). Derivation rules are not in the focus of this chapter, but are important when layering higher-level entailment regimes on top of the work presented in this chapter. Here, we use the Simple Interpretation. In a request rule, the head specifies an HTTP request. The request specification includes an

HTTP method, a request URI, and optionally a set of triple patterns to form the HTTP body (cf. Figure 5.1, where (3) dereferences an end of `sosa:hosts` links, and (4) changes the state of `saref:LightSwitch`es that fulfil a certain condition).

5.4.2. Intuition of the Semantics of the Condition-Action Rule Language

We informally give the operational semantics for the rule language by describing how an interpreter would execute the program in Figure 5.1: An interpreter evaluates the triple patterns of the body of the rules on RDF data that is either given initially (1) or downloaded as mandated by request rules with HTTP-GET requests (3). Meanwhile, the interpreter adds data as mandated by derivation rules (2). The interpreter executes derivations and HTTP-GET requests until it calculated the fixpoint. Last, the interpreter executes the HTTP-PUT, POST, DELETE requests of all request rules whose body holds in the data (4). The interpreter operates in a looped fashion. In this chapter, we give the rationale behind this operational semantics.

5.5. Related Work

In this section, we discuss related work subdivided by topic.

One-step updates of semantic data In Semantic Data Management, people studied updates to RDF Graphs, e. g. Magiridou et al. schema-preserving updates [151], and standardised updating the data in a triple store using SPARQL [81]. To update Linked Data using HTTP has been proposed in a Design Issues article [18] and detailed out in a W3C recommendation [196]. In previous work, Stadtmüller et al. proposed Linked Data-Fu, a language and an interpreter for interacting with Read-Write Linked Data [198]. Those works lack the notion of ASM steps.

Automation on the web Unlike automation systems such as van Kleek et al.'s [214], IFTTT[2], and Arktik[3], we do not assume centralised information and event processing, but decentralised information and rule evaluation on state information. Ripple by Shinavier [192], and Leinberger et al.'s work [148] allow for read-only scripting and programming with Linked Data. We also cover write operations. Unlike the multi-agent approach of Ciortea et al. [45], we work without a central platform for discovery and for distributing notifications.

Descriptive works We monitored, analysed (Chapter 4), and formally described (Chapter 3) the dynamics of Linked Data. Those works are insufficient to specify computation.

Web Service descriptions A focus of Semantic Web Services are service descriptions and the processing of such descriptions, with some work on operational semantics [5, 158, 38, 183], mainly used for analysing composed services, instead of execution.

[2] `http://ifttt.com/`, visited on 2018-06-10.
[3] `http://arktik.samsumg.com/`, visited on 2018-06-10.

```
# (1) Language feature: RDF:
<http://my-room.example/#it>
  sosa:hosts <http://t2-relay.example/#it> , <http://t2-climate.example/#it> .

# (2) Language feature: Derivations:
{ ?systemA sosa:hosts ?systemB . ?systemB sosa:hosts ?systemC . }
=> { ?systemA sosa:hosts ?systemC . } .

# Language feature: Conditional requests:
## (3) Defining how to retrieve world state (GET requests):
{ ?x sosa:hosts ?y . }
=> { [] http:mthd httpm:GET ; http:requestURI ?y . } .

## (4) Defining the logic (PUT, POST, DELETE requests):
{ <http://my-room.example/#it> sosa:hosts ?lightswitch , ?tempsensor .
  ?tempsensor a saref:TemperatureSensor ;
    rdf:value ?temperatureInCelsius .
  ?temperatureInCelsius math:lessThan 20 .
  ?light a saref:LightSwitch ;
    saref:hasState saref:Off .
  ?ir foaf:primaryTopicOf ?light . }
=> { [] http:mthd httpm:PUT ; http:requestURI ?ir ;
       http:body
         { ?light a saref:LightSwitch ;
             saref:hasState saref:On .
           ?ir foaf:primaryTopicOf ?light . } . } .

{ <http://my-room.example/#it> sosa:hosts ?lightswitch , ?tempsensor .
  ?tempsensor a saref:TemperatureSensor ;
    rdf:value ?temperatureInCelsius .
  ?temperatureInCelsius math:notLessThan 20 .
  ?light a saref:LightSwitch ;
    saref:hasState saref:On .
  ?ir foaf:primaryTopicOf ?light . }
=> { [] http:mthd httpm:PUT ; http:requestURI ?ir ;
       http:body
         { ?light a saref:LightSwitch ;
             saref:hasState saref:Off .
           ?ir foaf:primaryTopicOf ?light . } . } .
```

Figure 5.1.: Example for a simple rule program that turns on the lights in a room depending on the room temperature.

Descriptions/formalisations and HTTP Lately, descriptions in a Semantic Web Services fashion have gained traction again, especially on the Internet of Things [40], and even when working with HTTP [146, 171, 111]. The descriptions express in a standardised manner what you can do with a (server) API. Instead, we formally address user agent specifications. Our approach can be extended with planning (e. g. [217, 183]) if service descriptions are available. Thus, our Linked Data-based formalisation (which respects the HTTP message semantics) gives an alternative to approaches to service composition and execution that build on BPEL, an executable workflow language.

Application of ASMs in Semantic Technologies Before the advent of RDF and Linked Data, van Eck et al. [213] compared formalisms to specify dynamics for knowledge bases. Abstract State Machines (ASMs) stood out for their simplicity and ease of operationalisation. ASMs have also been used to describe the communication of services in a choreography in WSMO [183], a ontology to describe web services.

Combination of static and dynamic modelling Huang et al. [121] executed queries in linear temporal logic (LTL) over evolving ontologies, which is complementary to our approach. Similar to our work, Kifer [138] combined approaches for static and dynamic modelling for object-oriented modelling. While Graph Rewriting could serve as an alternative to ASM for the specification of the evolution of RDF Graphs, we think ASM is more compatible to the Semantic Web stack, as ASMs and Semantic Web languages are based on first-order logic.

5.6. Preliminaries

In this section, we extend our treatment of the foundational definitions of the technologies around Linked Data, which we covered in Chapter 2, with those aspects that are relevant to our proposed synthesis with Abstract State Machines. Moreover, we give the basic definitions of Abstract State Machines.

Specifically, we assume the following knowledge about technologies around Linked Data: Uniform Resource Identifiers (URIs) [21] for identifying things, the Hypertext Transfer Protocol (HTTP) [75] for interacting with things (with the HTTP message semantics in particular), the Resource Description Framework (RDF) [47] for static descriptions of things, and the RDF dataset to maintain descriptions of things from different sources. As the technologies are about the transfer and the description of state information, we first discuss the relations of different kinds of state (e. g. world and resource state). Next, we refresh the reader's knowledge of the model-theoretic semantics of RDF. We use model-theoretic semantics to describe static aspects in our formalisation. Last, we introduce Abstract State Machines (ASMs), on which we base our formalisation of dynamic aspects. If two approaches use the same term, we introduce subscripts for the origin ($\cdot_{\text{RDF}}$ vs. $\cdot_{\text{ASM}}$).

5.6.1. State

We distinguish different kinds of state: When successfully retrieving state using HTTP GET from a URI, we obtain a description the *resource's state*, e. g. the light in a room. The corresponding HTTP *request's state* can e. g. be "successful". We call the union of the state information about all resources *world state*, i. e. all resources in the world. An application (e. g. implemented as rule program) can maintain state (e. g. in writeable resources), which then represents the *application's state*, e. g. a note that something just happened. The application state is a subset of the world state. ASM terminology uses the term *system state*, which can be translated to our terminology as the subset of the world state that is relevant for an application, e. g. the building description and the weather report.

5.6.2. RDF Model-Theoretic Semantics

We use the graph-based data model Resource Description Framework (RDF) to describe things and their relations [47]. The RDF standard uses model theory to specify the meaning of an RDF Graph (e. g. that the URI for the light is interpreted as the resource that represents the light). We paraphrase the definitions as they are given in the corresponding W3C Recommendation [109].

Definition 9 (Interpretation$_{\text{RDF}}$, name, vocabulary$_{\text{RDF}}$, universe)**.** *An* interpretation *is defined as a mapping from $\mathcal{U}$ and $\mathcal{L}$ into some set called the* universe*. Both the mapping and the set can be constrained. A* name *is an element from the union $\mathcal{U} \cup \mathcal{L}$, i. e. URIs and Literals. A set of names is also called* vocabulary*.*

The basic interpretation for an RDF Graph is the *simple interpretation*. The simple interpretation defines the following mappings and constraints.

Definition 10 (Simple interpretation, extension)**.** *A* simple interpretation *is defined using the following sets and mappings: IR and IP denote the subsets of the universe that contain all resources and properties respectively. IEXT(p), called the* extension *of a property p, is a mapping $IP \to 2^{IR \times IR}$ that maps p to all pairs in IR that are connected by p. IS is a mapping $\mathcal{U} \to IR \cup IP$. IL is a partial mapping $\mathcal{L} \to IR$.*

To make the presentation succinct, we assume ground RDF Graphs in this thesis. The extension to Blank Nodes should be straight-forward.

5.6.3. Abstract State Machines (ASM)

For the dynamic aspects of our approach, we use Abstract State Machines (ASMs). Concretely, ASMs help us to define how to evolve Linked Data, which we interpret as RDF Dataset. Here, we give the standard definitions for ASMs to make the paper self-contained. The approach of ASMs has been developed towards the end of the 1980s with the aim to bridge between the specification and computation of programs [90]. ASMs are meant to improve on Turing's thesis. Turing Machines have proven too low-level to

specify algorithms larger than toy examples. In ASMs however, the level of abstraction can be adapted to the requirements of the use-case. ASMs have been extensively used to give operational semantics to programming languages including C, Prolog, and Java[4].

An ASM consists of an algebraic first-order structure and a set of transition rules on how to modify the structure. In the following, we give the basic definitions for ASMs.

Definition 11 (Vocabulary$_{\text{ASM}}$, function name, relation name, characteristic function name, variable, term)**.** *The* vocabulary Υ *be defined as a finite set of* function names *and their arity* $n \geq 0$*. The vocabulary also contains the nullary function names* `undef`*, and the boolean constants* `true` *and* `false`*. Moreover, the vocabulary contains the usual boolean operators as function names.* Relation names *and* characteristic function names *are function names for relations and characteristic functions (see Definition 12).* Terms *can be defined recursively:* Variables *are terms. If* f *is an* n*-ary function name and* $t_1, \ldots, t_n$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*

When considering terms, note the case of a nullary function name. For the presentation in this chapter, we characterise a function name with arity > 0 with their correctly sized argument list, e. g. to talk about an RDF Dataset for Linked Data in Section 5.7.2, we define $quad(\cdot, \cdot, \cdot, \cdot)$, which defines the function name *quad* with an arity of 4. We omit the argument list for nullary function names.

Definition 12 (Super-universe, function, relation, characteristic function)**.** *The* super-universe X *is a non-empty set.* X *contains e. g. all* functions $X^n \to X$, TRUE, FALSE, *and* UNDEF*. The boolean functions behave the usual way on* $\{\text{TRUE}, \text{FALSE}\}$*.* Relations *are functions that map* $X^n \to \{\text{TRUE}, \text{FALSE}\}$*.* A characteristic function *for a set is a function* $X \to \{\text{TRUE}, \text{FALSE}\}$ *that maps to* TRUE *iff the operand belongs to the set.*

To be able to visually tell the elements of Υ and X from each other, we use typewriter font for the nullary function names and small caps for their counterparts in the super-universe.

Definition 13 (Interpretation$_{\text{ASM}}$, transition rule, function update, static and dynamic function)**.** *An* interpretation $\mathcal{I} : \Upsilon \to X$ *maps the terms from an ASM vocabulary to a super-universe.* A transition rule R *is a function update, e. g.* $f(t_1, \ldots, t_n) := t_0$*, or a guarded function update, which has a boolean condition, e. g.* **if** *condition* **then** *function update(s).* A function update *changes the interpretation of a function name at given arguments.* A static function *is not subject to change by the function updates of an ASM, as opposed to a* dynamic function*.*

Definition 14 (Value and evaluated form)**.** *The* value *of a term from the vocabulary* Υ *is the term's referent in the super-universe* X *under the current interpretation* $\mathcal{I}$*. We denote the referent of a term* t *under an interpretation* $\mathcal{I}$ *using* $eval(t, \mathcal{I})$*. If* t *is a tuple, we have:* $eval(t, \mathcal{I}) := (eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))$*. If* t *is a function, we have:* $eval(f(t_1, \ldots, t_n), \mathcal{I}) := eval(f, \mathcal{I})(eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))$*.A function update in its*

[4]See the annotated ASM bibliography at `http://web.eecs.umich.edu/gasm/index.html`, visited on 2018-06-10.

so-called evaluated form *is* $eval(f(t_1, \ldots, t_n) := t_0, \mathcal{I}) = f(eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))) := eval(t_0, \mathcal{I})$.

Definition 15 (Algebra, state, run, step)**.** *A (static)* algebra *or* state *is a triple* $(\Upsilon, X, \mathcal{I})$ *of a vocabulary* Υ*, a super-universe* X*, and an interpretation* $\mathcal{I}$*. A* step *is a transition from one state to the next by the means of firing transition rules. A* run *is a sequence of steps.*

The transition rules to be fired in one step are composed in an update set.

Definition 16 (Update set)**.** *The* update set *to be fired in an interpretation* $\mathcal{I}$ *under the transition rule set* T *can be defined as* $updates(\mathcal{I}, T) := \{eval(u, \mathcal{I}) | u \in T \wedge$ (*the condition of* u *holds in* $\mathcal{I} \vee u$ *has no condition*)$\}$

In ASM, we assume discrete time. Time progresses from one state to the next, i. e. in a run, there is an ordered sequence of system states. There may be multiple transition rules that fire in one state. If multiple transition rules want to update the interpretation of a function name for the same arguments, then the system halts because of the conflict.

Definition 17 (Abstract State Machine)**.** *An* Abstract State Machine *(ASM) can be defined as a quadruple of a vocabulary* Υ*, a super-universe* X*, an interpretation* $\mathcal{I}_0$ *for time* t_0*, and a set of transition rules* T*:* $ASM := (\Upsilon, X, \mathcal{I}_0, T)$

We now introduce variables to be able to range over names in conditions and updates. In ASMs, we have to state from which set of individuals those individuals are to take that are to be bound to variables. Therefore, a variable declaration is of the following form:

Variable(s) `?varname` **range(s) over** *set*
End variable scope

Formally, the variables can be covered by the help of an auxiliary vocabulary that contains the variables as nullary function names [90].

5.7. Abstract State Machines and Linked Data + Rules

This section is about our proposed synthesis of ASM, and Linked Data + Rules with the help of RDF model theory and HTTP semantics. We first describe on a high level the commonalities between ASM and RDF model theory, which motivate our synthesis, and then the different foci of both approaches. After that, we provide a synthesis of ASM and Linked Data + Rules. While the formalisation has the user agent in the focus, which regards Linked Data as external functions, we next sketch how a server fits into the picture. Then, we sketch how the synthesis can be used for specifying computation. Subsequently, we use the synthesis to give semantics to a rule language and discuss our findings. Last, we derive requirements for a Linked Data user agent specification language.

5.7.1. Overview

While both, ASMs and RDF, are defined in terms of interpretations of vocabularies into a (super-)universe, the focus of each approach is different: RDF model theory is about whether different RDF Graphs entail each other or whether models can exist for an RDF Graph. Therefore, conditions on the universe are in the focus. The interpretation is static in RDF model theory. ASMs are all about the evolution of system state, which manifests itself in the evolution of the interpretation of a vocabulary into a super-universe. Therefore, the transition function T is in the focus, in which updates to the interpretation are stated and the conditions under which the updates happen. Linked Data is about the publication and the updating of RDF data on the web using HTTP, where the RDF describes the state of a resource.

The high-level idea of the synthesis, which is described in the subsections of Section 5.7.2, is to (1) define RDF model theory for Linked Data using RDF Datasets, (2) define ASM functions $statement(\cdot,\cdot,\cdot)$ and $quad(\cdot,\cdot,\cdot,\cdot)$ for Linked Data/RDF Datasets and the functions' interpretations, (3) define an ASM transition function for Linked Data/RDF Datasets using rules with HTTP requests in the head (4) define how the ASM evaluation of the $statement(\cdot,\cdot,\cdot)$ function and the ASM updates to the $quad(\cdot,\cdot,\cdot,\cdot)$ function can be done in accordance with the semantics of HTTP requests, and (5) use above definitions in the definition of an ASM.

5.7.2. Synthesis

In this section, we take the definitions from the simple interpretation from RDF model theory and use them, slightly amended, for RDF Datasets, in the definition of an ASM. The synthesis allows us to specify the evolution of an RDF Dataset using ASM semantics. Note that the terms *vocabulary* and *interpretation* used in ASM and RDF model theory mean the same. The term *universe* in RDF maps to the term *super-universe* in ASM.

Define RDF model theory for Linked Data using RDF Datasets

We base our considerations on two semantics for RDF Datasets that have been discussed in Sections 3.2 and 3.5 of a note of the W3C RDF Working Group [225].

We start out with definitions to simplify our presentation: Like the W3C Recommendation on model-theoretic semantics for RDF [109], we use as vocabulary the infinite sets of all URIs $\mathcal{U}$ and Literals $\mathcal{L}$ in our definitions, in contrast to the fixed vocabulary used by the Linked Data sources under consideration. Correspondingly, IS and IL interpret all $\mathcal{U}$ and $\mathcal{L}$, and IR has corresponding elements. We point to [109] for how to amend the definitions to finite vocabularies. To further ease the presentation, we assume ground graphs. Moreover, be $IP = IR$, such that we only need to address different $IEXT$ in our considerations. When working with RDF Datasets, we assume that the same terms in the vocabularies of the different graphs have the same referent when interpreted using IS and IL. We do not consider redirects in the definitions, although they can be layered on top by defining a function for the correspondence between a non-information resource and an information resource. In the definitions, we also assume all Linked Data sources

to be in $\mathcal{N}$, the named graphs of our RDF Dataset, i. e. we assume web-completeness in the terminology of [101]. By restricting $\mathcal{N}$, we can implement other completeness classes.

Of the W3C RDF WG Note on RDF Dataset semantics [225], two sections are of particular use for our considerations:

3.5 "Named graph[s] are in a particular relationship with what the graph name dereferences to" defines an interpretation for each graph in the RDF Dataset, hence we can define an extension function per dereferenceable source

$$IEXT_c := \textit{Extension function of the graph available at } \mathrm{c}$$

3.2 "Default graph as union or as merge" defines the default graph of an RDF Dataset as RDF union or merge of all named graphs in the RDF Dataset. As we assume ground graphs, there is no difference in RDF union or merge. Hence, we can define the extension function of the union as:

$$IEXT^{\text{UNION}}(p) := \bigcup_{c \in \mathcal{N}} IEXT_c(p)$$

The default graph can be regarded as knowledge of the client, e. g. knowledge that is not available at any $n \in \mathcal{N}$. Derivations also go into the default graph. The default graph can also be used in the case of higher-level entailment regimes for stating axiomatic triples in case they cannot be dereferenced from a Linked Data source.

Define ASM functions quad$(\cdot,\cdot,\cdot,\cdot)$ and statement$(\cdot,\cdot,\cdot)$ for Linked Data/RDF Datasets and the functions' interpretations

We define

$$quad(s,p,o,c) : IR \times IR \times IR \times \mathcal{N} \to \{\text{TRUE}, \text{UNDEF}\}$$

as the characteristic ASM function for $IEXT_c$. Moreover, be

$$statement(s,p,o) : IR \times IR \times IR \to \{\text{TRUE}, \text{UNDEF}\}$$

the characteristic ASM function for $IEXT^{\text{UNION}}$. The *statement* function has also been used in [193] to process RDF using Prolog.

So far, we have defined Linked Data as static ASM functions. Now, we proceed by introducing dynamics.

Define how the ASM transition function for Linked Data/RDF Datasets can be stated

The rules can be given as usual in ASM in rules of the form **if** *condition* **then** *update(s)* modifying the interpretations of function names. We use conjunctions of the $statement(\cdot,\cdot,\cdot)$ function for the conditions (reflecting BGP queries from SPARQL [93]) and the conjunctions of the $quad(\cdot,\cdot,\cdot,\cdot)$ function for the updates. Both the conditions and updates can contain variables.

Define how the ASM evaluation of the statement$(\cdot,\cdot,\cdot)$ function and the ASM updates to the quad$(\cdot,\cdot,\cdot,\cdot)$ function can be done in accordance with the semantics of HTTP requests

For the evaluation of the conditions, we conceptually consider $statement(\cdot,\cdot,\cdot)$ an external function in ASM. When evaluating the function $statement(\cdot,\cdot,\cdot)$, we make HTTP-GET requests to all $n \in \mathcal{N}$ and evaluate the graph patterns on the data thus received. For the default graph, no request needs to be made. The updates are done using the $quad(\cdot,\cdot,\cdot,\cdot)$ function, which takes an additional c as parameter. The updates correspond to HTTP-PUT, POST, DELETE requests.

We perform the requests ordered in accordance with ASM steps, i. e. we first evaluate all HTTP-GET requests, and collect the updates that are mandated by the transition functions. After we collected the updates to be performed, we carry out the corresponding requests in bulk, i. e. multiple HTTP PUT, POST, DELETE requests at a time. After all updates in the form of HTTP-PUT, POST, DELETE requests have been performed, we continue with the next ASM step. The result of the HTTP-PUT, POST, DELETE requests is dependent on server implementations, see Section 5.7.3.

Define the ASM

We next define the ASM for Linked Data. We follow Definition 17, and detail what the vocabulary, the super-universe, and the interpretation of the ASM are in Linked Data. The transition function can then be defined as usual in ASM. We present the analogy to Notation3 in 5.7.4.

$$ASM := (\Upsilon, X, \mathcal{I}_0, T)$$

The vocabulary Υ contains all RDF names (all URIs and Literals), the boolean operator $\wedge$, and constant names. We omit (1) the boolean name `false` and the boolean operator $\neg$ because of the open world assumption, which is typically made on the Semantic Web [113], and (2) the operator $\vee$, because in a BGP, conditions are only connected using conjunctions. Disjunctions can be stated by using multiple rules. We add the function names *statement* and *quad*:

$$\Upsilon := \mathcal{U} \cup \mathcal{L} \cup \{\texttt{true}, \texttt{undef}\} \cup \{\wedge\} \cup \{quad, statement\}$$

The super-universe X is a set that contains elements for all RDF Graphs, and functions $X^n \to X$ including the boolean *and*.

$$X := IR \cup IP \cup \{\text{TRUE}, \text{UNDEF}\} \cup \{f | f : X^n \to X\}$$

The interpretation $\mathcal{I}_t$ of an element y of the vocabulary Υ at time t, consists of $IS(\cdot)$ and $IL(\cdot)$ from RDF model-theoretic semantics for the RDF names, mappings for the boolean

`true`, and `undef`, and functions for $quad(\cdot,\cdot,\cdot,\cdot)$ and $statement(\cdot,\cdot,\cdot)$, and $\wedge$:

$$\mathcal{I}_t(y) := \begin{cases} IS(y) & \text{if } y \in \mathcal{U} \\ IL(y) & \text{if } y \in \mathcal{L} \\ \text{TRUE} & \text{if } y = \texttt{true} \\ \text{UNDEF} & \text{if } y = \texttt{undef} \\ \in \{f | f : X^n \to X\} & \text{if } y \in \{quad, statement, \wedge\} \end{cases}$$

5.7.3. Linked Data Servers

Although our formalisation is made for the user agent, we show for completeness the server behaviour, which reflects the update of the $quad(\cdot,\cdot,\cdot,\cdot)$ function for HTTP-PUT, POST, and DELETE requests: A server processing an HTTP-PUT request to the target information resource `http://t2-relay.example/modules/relay/1` with the body

```
<> foaf:primaryTopic <#it> .
<#it> a saref:LightSwitch ;
  saref:hasState saref:On.
```

sets

$$quad(\cdot,\cdot,\cdot,\texttt{http://t2-relay.example/modules/relay/1})$$

to `true` for the arguments

$$(\texttt{<>}, \texttt{foaf:primaryTopic}, \texttt{<\#it>})$$

$$(\texttt{<\#it>}, \texttt{a}, \texttt{saref:LightSwitch})$$

$$(\texttt{<\#it>}, \texttt{saref:hasState}, \texttt{saref:On})$$

and `undef` for all other triples. A server processing an HTTP-POST request with the target of a LDP container, e. g. `http://ldpc.example/` and the RDF payload *payl* first determines a new URI *sub* that is subordinate to the container resource, then defines a new graph by relativising all URIs in the *payl* against *sub* and PUTs the new graph at *sub* into the RDF Dataset. Moreover, a membership triple for *sub* is added to the representation of the container resource.

5.7.4. Operational Semantics for the Condition-Action Rule Language

In this section, we give operational semantics to our condition-action rule language using the synthesis. We use the program in Figure 5.1 as example: The program consists of RDF (1), derivations (2), and requests (3+4), the latter to interact with the environment. Remember that in an ASM step, updates are executed after the update set has been constructed entirely. To construct the update set, all rules have to be evaluated on the current state.

In our language, information about the current system state is given by RDF, by evaluating all derivations, and by evaluating all GET requests on the Linked Data

sources until the fixpoint has been calculated. In our example, we know from the RDF (1) that the room `http://my-room.example/#it` has a relation of type `sosa:hosts` to `http://t2-relay.example/#it` and `http://t2-climate.example/#it`. The presence of a triple with predicate `sosa:hosts` triggers the conditional HTTP-GET request (3). The information obtained using the HTTP-GET request again contains triples with `sosa:hosts` predicate (see Example 5). The derivation rule (2) derives triples based on the transitivity of the `sosa:hosts` predicate. Hence, we know that the room not just hosts the two devices, but also the sensors and actuators. After no further knowledge can be derived using rules, the fixpoint has been calculated. The updates in our language are the unsafe requests. The conditional PUT, POST, DELETE requests (4) are collected during the information gathering about the current state, and are executed after the fixpoint calculation. Note that using conditional requests, we can do hypermedia-style link following. In our example, the condition for the last rule holds in the knowledge after the fixpoint calculation. Hence, the light is turned on or off depending on the temperature.

To execute programs in our language along the ASM step semantics, an interpreter operates in nested loops (see Figure 5.2 for its algorithm). The algorithm takes as input a rule program and emits a sequence of sets of safe and unsafe requests, depending on the available state information. Note that ASM steps in combination with GET requests implement polling.

5.7.5. Discussion: Computation, ASMs, Simple Reflex Agents, and Linked Data

ASMs are a model of computation, i. e. one can specify arbitrary computation using the model, e. g. algorithms and programs. In the previous section, we defined an ASM view on Linked Data. Hence, we now can specify arbitrary computation with the state of the computation represented in Linked Data. Our ASM-based formalisation of Linked Data is based on the assumption that we have exclusive read and write access to all relevant URIs during an ASM step. When generalising to a setting with multiple agents, ASMs typically assume circulating exclusive access to the system state. As there is no central control in a web setting, we cannot circulate the access right. Thus, there is no general solution to the problem of concurrent access, but a number of remedies: (1) in a realistic setting, we do not have the entire Linked Data in our system state (i. e. web-completeness in the terminology of [101]), but a considerably smaller subset. (2) on the web, most access is typically reading, and only little is writing [173]. (3) if the servers use ETags [74], a client can send conditionally writing requests, which fail if a concurrent update has happened. (4) with sufficiently fast data processing, the problem can be mitigated.

Using our ASM view on Linked Data, we can also implement simple reflex agents for Linked Data, the simplest agent type in Russell and Norvig [185]. Simple reflex agents perceive the environment and choose their action by matching condition-action rules on the perceived environment. Then, the agent carries out the action and repeats. This sense-act cycle aligns with an ASM step of first, matching rules and second, acting in the

```
Require: assertions                          ▷ Triples to be asserted in every ASM step
Require: rules                                              ▷ Derivation and request rules
  var unsafeRequests: set⟨request⟩
  var data, oldData: set⟨triple⟩
  var fixpointReached: boolean
  while true do                                                  ▷ Loop of the ASM steps
    unsafeRequests.clear()
    data.clear()
    data.add(assertions)
    repeat                                          ▷ Loop for determining the update set
      fixpointReached ← true
      for rule : rules do
        if rule.matches(data) then                              ▷ Also empty rule bodies
          oldData = data.copy()
          if rule.type==derivation then
            data.add(rule.match(data).data)
          else                                  ▷ So the rule must be an interaction rule
            if rule.match(data).request.type==GET then
              data.add(rule.match(data).request.execute())
            else
              unsafeRequests.add(rule.match(data).request)
            end if
          end if
          if ! data.copy().remove(oldData).isEmpty() then
            fixpointReached ← false
          end if
        end if
      end for
    until fixpointReached
    for request : unsafeRequests do                          ▷ Enacting the update set
      request.execute()
    end for
  end while
```

Figure 5.2.: The nested loops for the ASM-based operational semantics for the rule language.

form of updates.

5.7.6. Requirements for a Linked Data User Agent Specification Language

What current languages to specify queries and updates are missing is a way of expressing the transition function T. In ASM, T is given using rules in the form **if** *condition* **then** *function update(s)*. T is then executed in ASM steps. Thus, we need a language

1. that allows to express conditions on state information
2. that allows to define updates that send state information,
3. whose semantics adhere to ASM steps, i. e. one repeatedly first evaluates all conditions of all rules and collects all updates, and then one evaluates all updates in bulk.

We chose N3 as rule language in the context of RDF data and gave ASM-based semantics to N3. Other approaches including SPARQL updates [81] or RUL [151] as very elaborate ways to address point 1. They do not implement server interaction using exchange of state representation, but RPC-style interaction (point 2). They also do not implement bulk updates, as required by ASM steps (point 3).

5.8. Evaluation

In this section, we provide a formal evaluation by studying the expressivity of the approach and an experimental evaluation using a synthetic benchmark in the building automation domain.

5.8.1. Formal Evaluation

To show the expressiveness of our approach, we implemented a Turing Machine in ASM4LD. A Turing Machine is typically defined as a septuple of a set of machine states, a tape alphabet, the blank symbol as part of the tape alphabet, an input alphabet, the initial state, the set of final states, and a transition function [118]. For our considerations, we omit the alphabets and the machine states states and assume them as given implicitly in the transition function.

Our Turing Machine operates on a tape that is deployed as Read-Write Linked Data. The tape is described as a linked list using the RDF List vocabulary. However, we use URIs of writeable Linked Data resources instead of blank nodes to describe the list, as follows. For instance, the (finite) tape with element zero followed by one followed by zero can be described using the Turtle list syntax as `( 0 1 0 )`. The corresponding RDF graph contains the following triples (given in Turtle without using the list syntax):

```
_:bn1 rdf:first 0 ;
      rdf:rest _:bn2 .
_:bn2 rdf:first 1 ;
```

```
      rdf:rest _:bn3 .
_:bn3 rdf:first 0 ;
      rdf:rest rdf:nil .
```

We divide the triples of the RDF Graph by the subject, and create URIs of writeable Linked Data resources for the blank nodes. The graphs obtained when dereferencing the URIs consist of the triples with the blank nodes substituted by the URIs. The result of the processing of the RDF list can be described using the following RDF dataset:

Name	Graph
http://ldpc.example/ldpr1	<ldpr1> rdf:first 0 ; rdf:rest <ldpr2> .
http://ldpc.example/ldpr2	<ldpr2> rdf:first 1 ; rdf:rest <ldpr3> .
http://ldpc.example/ldpr3	<ldpr3> rdf:first 0 ; rdf:rest rdf:nil .

Hence, we can write to the individual list elements (i. e. the tape) using HTTP-PUT requests.

The transition function of a Turing Machine is typically given as a set of quintuples of current state, scanned symbol on the tape, symbol to be printed, the direction of the head, and the next state. We give the transition function as an RDF graph available as Linked Data, where one quintuple can be written as follows:

```
@prefix tm:  <http://www.example.org/turing-machine#> .

[] tm:hasCurrentTapeSymbol 0 ;
   tm:hasCurrentState <q0> ;
   tm:hasNextState <q0> ;
   tm:hasDisplacement tm:R ;
   tm:hasWriteSymbol 0 .
```

Last, we have to give the state of the Turing Machine, which we give as writeable Linked Data:

```
@prefix tm: <http://www.example.org/turing-machine#> .

<> tm:hasInitialPosition <ldpr2> ;
   tm:hasCurrentPosition <ldpr2> ;
   tm:hasCurrentState <q0> .
```

Then, we can run the Turing Machine using four Linked Data-Fu rules with ASM4LD semantics, see Appendix B. While the Turing-completeness is, of course, an inherent property of the ASM part of the approach, it persists despite the restrictions we imposed to make ASM fit Linked Data. All parts of our approach are necessary: Read-Write Linked Data for tape and machine state, and request rules executed in ASM steps.

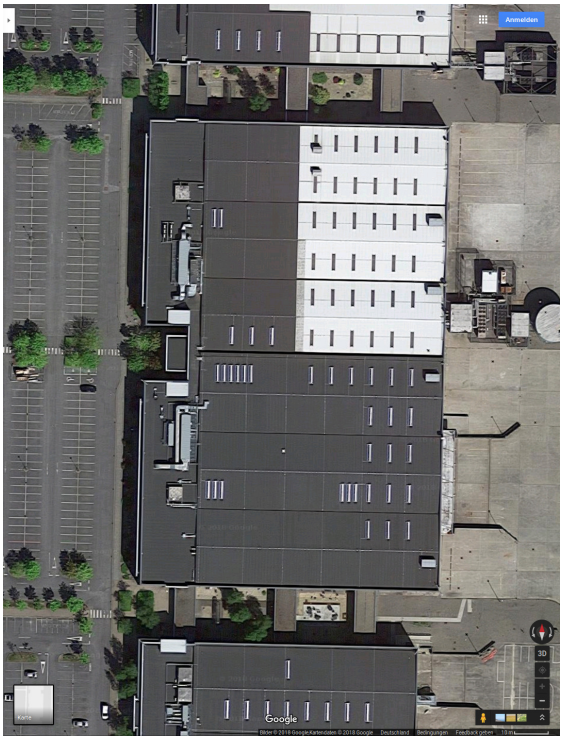


Figure 5.3.: Building 3 of IBM Research Dublin. Image from Google Maps.

5.8.2. Experimental Evaluation

To describe our empirical evaluation, we start out with defining the data and different scenarios for data access. Next, we present different workloads. Then, we describe the experimental setup. Next, we present and discuss our results. Last, we showcase applications of our approach.

Data

While we used two devices from the Internet of Things in the examples of this thesis, we use a scaled-up version as the scenario of our experiments. Specifically, we use examples from the Smart Building domain, where also lights and temperature sensors are deployed, inspired by recent work in Building Management Systems: NIST identified interoperability as a major challenge for the building industry [79]. To raise interoperability in Building Management Systems, Balaji et al. developed Brick [11], an ontology to model buildings and corresponding building management systems.

We use a description of building 3 at IBM Research Dublin in Ireland, see Figure 5.3. Balaji et al. provide a static description of this building using the Brick ontology[5]. The

[5] `https://github.com/BuildSysUniformMetadata/GroundTruth/blob/2e48662/building_instances/IBM_B3.ttl`, visited on 2018-06-10.

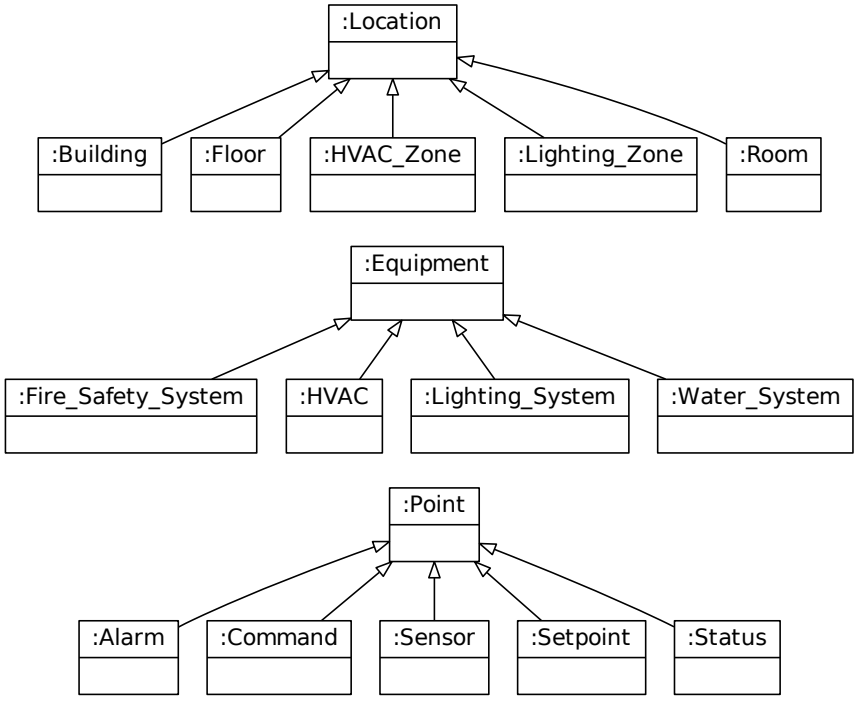


Figure 5.4.: Excerpts from the Brick ontology as UML Class Diagram. UML class inheritance denotes RDFS sub class relations. The empty prefix expands to `http://buildsys.org/ontologies/Brick#`.

description covers the building and the building's parts (e. g. rooms) and the different parts of the building's systems (e. g. lights and switches). As there is nothing to automate in the static description, we introduce dynamics by adding information about the state of these system parts (e. g. on/off) in the form of properties from the SSN ontology [92]. We serve those properties as writeable Linked Data.

In the example of Section 5.4, we use ASM4LD to program an user agent that controls one light in the building in a straight-forward fashion (turn the light off or on depending on the room temperature). For the evaluation, we use more complex control schemes with up to 30 rules and control the lights of the entire building based on time of day, a weather API and luminance sensor values. We use the building 3's lighting subsystem, as lamps and switches can be regarded as having a discrete state. We use this particular building, as the provided building description includes a lighting subsystem.

The core classes of the Brick ontology are depicted in Figure 5.4. The first part of the ontology, `Location` and subclasses, allows for describing a building in terms of the subdivisions provided by the brickwork, e. g. floors and rooms. The second part of the ontology, `Equipment` and subclasses, allows for describing different systems that run the building, where we focus on the lighting system. Other systems include water, and HVAC (heating, ventilation, and air conditioning). The third part of the ontology, `Point` and subclasses, allow to talk about sensors and actuators in the building.

Building 3 is box-shaped with its long side oriented from north to south, see Figure 5.3. In the RDF description, the building is subdivided into floors and wings. The rooms are assigned to their corresponding floor and wing (also has-part relationships). The lights are directly assigned to rooms or wings. A light system can consist in (1) an occupancy sensor that determines whether there are people in its vicinity, (2) a luminance command, in other words, a switch to control the lights, and (3) a luminance sensor that we consider to be triggered by daylight. We provide basic statistics in Table 5.1.

We bring the static building description "to life" by adding `ssn:hasProperty` links to dynamic Linked Data resources representing switches, occupancy sensors, luminance sensors, and lights. The state of each resource relevant for the evaluation is on/off for the lights, and a numeric value for the luminance sensors.

Workload

In our evaluation, we implement different user agents that provide automation. We run the rules on different parts of the buildings to investigate how the approach scales. In terms of data access, i. e. the evaluation of the *quad* function, there are two extremes:

D1 The whole building description is available as one file, which is loaded from one single Linked Data source on the network

D2 The building description is available subdivided into one Linked Data source per resource in the building. The Linked Data sources can thus be accessed as required following links. We built the Linked Data representation as follows: We subdivided the description into one-hop RDF Graphs around each URI from the building and provide each graph for dereferencing at the corresponding URI. No data is lost in the subdivision, as there are no blank nodes in the description.

We scale the scenario from one room to the whole building.

In terms of automation workloads, we scale both the rule complexity, and energy efficiency and comfort of the building. According to UNEP [209], lighting is the second-highest energy consumer in commercial buildings, optimised control can yield substantial savings.

W1 Turn all light switches on (no conditions / baseline; 9 rules)

W2 Working hours (conditions and another source, a Linked Data clock / clock-based straight-forward control; 46 rules): The lights are on per default during working hours.

W3 Weather API (more complex rules, another source / raising energy efficiency; 20 rules): We turn the lights on only if a lack of sunlight indicates illumination.

W4 Luminance sensor (numerical computations / further raising efficiency; 34 rules): We consider luminance sensor values in the rooms to determine whether the lights should be on.

Table 5.1.: Basic counts for building 3 and the benchmark.

Rooms	281
Floors	2
Wings	3
Lights	166
Lights w/occupancy sensors	156
Lights w/luminance commands	126
Lights w/luminance sensors	60
Lights w/all three	48
Triples in IBM_B3.ttl	24 947
Resources in the LDP Container	3 281
Dynamic resources	551

W5 Luminance sensor w/room-individual thresholds (more complex computation / raising individual comfort; 19 rules): We assign an individual light threshold per room.

Experimental Setup

We use the engine Linked Data-Fu version 0.9.12[6] to run the rule programs. We use LDBBC version 0.0.6[7] as LDP Container to serve the static building data. We implement the following dynamic sources: (1) an RDF weather API built from a sample file from OpenWeatherMap[8], a JSON-LD context, and a sunset/sunrise simulation for Dublin, and (2) luminance sensor readings according to time of day and season. We run the experiments on a laptop with an Intel Core i7-5600U CPU, 12 GB of RAM, and Ubuntu Linux 17.04. After the dynamic sources simulated one year, i. e. one minute in wall-clock time, we stop one experiment leading to repetition counts of up to 7'500 depending on the runtime of one ASM step. The evaluation system including data and rules can be found online[9].

Results and Discussion

We present the results for D1 in Table 5.3a and for D2 in Table 5.3b. We report median values as we observed that when firing requests with high frequency, there are high outliers in the 99th percentile, which spoil mean and standard deviation. The median, however, is stable. The order of magnitude of the measurements in varies in Table 5.3b, in contrast to Table 5.3a. This is owed to the fact that each ASM step, regardless of the data that is actually needed, the whole building description has to be transferred in

[6] `http://linked-data-fu.github.io/`, visited on 2018-06-10.
[7] `http://github.com/kaefer3000/ldbbc`, visited on 2018-06-10.
[8] `http://www.openweathermap.org/`, visited on 2018-06-10.
[9] `http://github.com/kaefer3000/rwld-brick-benchmark`, visited on 2018-06-10.

Table 5.2.: Median times [ms] for one ASM step.

Rooms	W1	W2	W3	W4	W5
1	484	572	510	554	561
5	480	582	501	574	582
10	498	584	529	605	618
20	537	631	562	719	687
First Floor	563	629	590	750	728
Wing 42	527	595	550	651	604
Building 3	605	734	613	794	788

(a) In scenario D1, data access from one single Linked Data source.

W1	W2	W3	W4	W5
8	8	8	8	8
40	38	38	40	40
85	80	79	88	88
259	238	228	320	268
938	1 690	891	1 063	1 048
1 435	1 427	1 371	1 664	1 408
2 442	2 187	2 192	2 542	2 497

(b) In scenario D2, data access via Linked Data + link following.

D1, which is 2.3 MB per transfer. Compared to that, the link following approach D2 can access the building data as required. This fine granularity comes at the price of additional requests, which make the processing of the whole building more expensive. The drop in time between W2 and W3 can be explained by the reasoning that is employed in W2 to check whether the current day and hour is a working hour, which is not needed in W3. The increase between W3 and W4 underestimates the real cost of the computations, as we go from all lights in W3 to only those lights with sensors in W4, cf. Table 5.1. The drop between W4 to W5 can be explained by the smaller number of lights with luminance sensors. The rise in time for the workloads in Table 5.3a is due to that different workloads need to access different amounts of data.

Applicability

We give two examples of how we applied our approach.

Composition of RESTful services to VR systems We used our approach to connect different parts of Virtual Reality systems: In the i-VISION project, we connected a flight simulator to a workflow analysis software [132] (ca. 90 rules). In a demo [137], we connected a Microsoft Kinect sensor and APIs from the Web to a 3D engine (ca. 60 rules), where the interpreter running the rules performed about 30 ASM steps per second (i. e. the refresh rate of the Kinect sensor).

Specification of Operational Semantics Similar to previous applications of ASMs to define the operational semantics of programming languages, we can use our approach to define the operational semantics of a workflow language using our approach (ca. 30 rules), see the Chapter 6.

5.9. Conclusion and Future Work

We have presented a formal approach for capturing the dynamics of Linked Data with the aim of specifying user agents. To this end, we gave a synthesis of RDF model theory, ASMs, and HTTP, and described requirements for implementing the synthesis. We applied the approach to give semantics to a rule language such that we can specify Linked Data user agents. We presented application scenarios and showed our approach to be Turing complete. We evaluated our approach in a Smart Building setting.

In the paper, we took the first step towards autonomous agents that operate on Linked Data: We presented a formal basis for the execution of agents. Still, there are more steps to take, a formal notion of (1) internal state in the agent, and (2) a notion of goals and capabilities, which we will address in future work.

Yet, we believe our work can already be applied today: Although experts built the applications in Section 5.8.2, the experiments of van Kleek et al. [214] or the success of IFTTT [210] show that rules are a way of programming that is highly relevant also for end-users.

6. Workflows in Linked Data

Parts of this chapter have been published in:
▷ Tobias Käfer and Andreas Harth. "Specifying, Monitoring, and Executing Workflows in Linked Data Environments". In: *Proceedings of the 17th International Semantic Web Conference (ISWC)*. 2018, pp. 424–440.

Using the model of computation of the previous chapter, we can specify computation in rules in the context of Linked Data. Hence, we can compose applications from components that provide Linked Data interfaces and specify the application behaviour in rules. Yet, it is workflows, which have been identified as paradigm to specify behaviour that is suitable for in integration scenarios by Jablonski and Bussler [125], easy to understand (for validation and specification by humans), and formal (for execution and verification by machines). E. g., consider an evacuation support workflow for a smart building (cf. task 4 in our evaluation, Section 6.8.3), which integrates multiple systems of the building, should be validated by the building management and the fire brigade, verified to be deadlock-free, and executable. Hence, we address the research question: *How to specify, monitor, and execute applications given as workflows in the environment of Read-Write Linked Data?*

Example 10. *Imagine the Internet of Things devices from our example. Using a (admittedly very simple) workflow, we can specify that two lights should be turned on sequentially, see Figure 6.1. Our approach also works in complex scenarios, which we consider in our evaluation, see Section 6.8.*

6.1. Challenges

To use workflows in the environment of Read-Write Linked Data is difficult as the environment is fundamentally different from traditional environments where workflows are used. Elmroth et al. argue that the properties of the environment determine the model of computation, which serves as the basis of a workflow language [61]. Consequently, we have developed ASM4LD, see Chapter 5, a model of computation for the environment of Read-Write Linked Data. In this chapter, we investigate an approach for a workflow language consisting of an ontology and operational semantics in ASM4LD. In the investigation of the approach, the differences between traditional environments of workflow languages and the environment of Read-Write Linked Data (i. e. RDF and REST) pose challenges:

Querying and reasoning under the open-world assumption [181] Ontology languages around RDF such as RDFS and OWL make the open-world assumption (OWA) [181]. However, approaches from workflow management operate on relational databases,

which make the CWA. Closedness allows e. g. to test if something holds for *all* parts of a workflow.

The absence of events in REST HTTP implements CRUD (the operations create, read, update, delete), but not the subscriptions to events. However, approaches from workflow management use events as change notifications.

While both challenges could be mitigated by introducing assumptions (e. g. negation-as-failure once we reach a certain completeness class [101]) or by extending the technologies (e. g. implement events using Web Sockets [71] or Linked Data Notifications [34]), those mitigation strategies would restrict the generality of the approach, i. e. we would have to exclude components that provide Linked Data, but do not share the assumptions or extensions of the mitigation strategy.

6.2. Contributions

The two main contribution of this chapter are as follows:

- An ontology to specify workflows and workflow instances modelled in OWL LD [85] (Section 6.5), which allows for querying and reasoning over workflows and workflow instances under the OWA. The ontology is strongly related to the standard graphical workflow notation, BPMN, via the workflow patterns [211].
- An operational semantics for our workflow ontology. The operational semantics allows for both, monitoring and executing workflows. We use ASM4LD, a model of computation for Read-Write Linked Data in the form of a condition-action rule language (Section 6.7), which does not require event data and is directly executable. We maintain workflow state in an LDP container.

Fast data processing thanks to OWL LD and the executability of ASM4LD allow to directly apply our approach in practice. In the evaluation (Section 6.8), we present a Virtual Reality showcase, and a benchmark in an Internet of Things setting. We also show correctness and completeness of our approach. We conclude in Section 6.9.

6.3. Related Work

We now survey related work grouped by field of research.

Workflow Management Previous work in the context of workflow languages and workflow management systems is based on event-condition-action (ECA) rules, whereas our approach is built for REST, and thus works without events. This ECA rule-based approach has been used to give operational semantics to workflow languages [122], and to implement workflow management systems [35]. Similar to the case handling paradigm [212], we employ state machines for the activities of a workflow instance.

Web Services WS-*-based approaches assume arbitrary operations, whereas our approach works with REST resources, where the set of operations is constrained. The BPM community has compared WS-*- and REST-based approaches [174, 226]. Pautasso et al. proposed extensions to BPEL such that e. g. a BPEL process [170] can invoke REST services, and that REST resources representing processes push events [172]. While those extensions make isolated REST calls fit the Web Services processing model of process variable assignments, we propose a processing model based on integrated polled state.

Semantic Web Services We can only present a selection of the large body of research conducted in the area of SWS. Approaches like OWL-S, WSMO and semantic approaches to scientific workflows like [84] are mainly concerned with service descriptions and corresponding reasoning for composition and provenance tracking. In contrast to our work, which is based on REST, SWS build on Web Service technology for workflow execution, e. g. the execution in the context of WSMO, WSMX [91], is entirely event-based. European projects such as "Super" and "Adaptive Services Grid" build on WSMO.

Ontologies for Workflows Similar to workflows in our ontology, processes in OWL-S are also tree-structured (see Section 6.5) and use lists in RDF. Unlike OWL-S, our ontology also covers workflow instances. Rospocher et al. [184] and the project "Super" developed ontologies that describe process metamodels such as BPMN, BPEL, and EPC[1]. Their ontologies require more expressive (OWL) reasoning or do not allow for execution under the OWA.

6.4. Preliminaries

We assume that the reader is familiar with Read-Write Linked Data and ASM4LD, i. e. Chapter 5. Again, we remark on the notion of state.

State

In our remarks on the notion of state in Section 5.6.1, we distinguished a *resource's state*, which can be obtained using one HTTP-GET request, the *world's state*, which is the union of the state of all resources, and an *application's state*, which is the subset of the world's state where an application maintains the application's state. In this chapter, applications are defined as workflows. Our application state is thus the state of a workflow instance, which we maintain in writeable Linked Data. Where in the Chapter 5 on ASM4LD, the application logic was, generally speaking, stateless and given in rules, in this chapter, the application logic is indeed stateful as the workflow instance state (e. g. which activities are active, which are done) is core to the application.

[1] `http://www.ip-super.org/content/view/129/136/`, available in the Web Archive as of 2018-06-10.

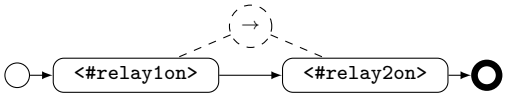


Figure 6.1.: Workflow (solid: BPMN notation) with sequential activities (`<#relay1on>`, `<#relay2on>`). Dashed: the tree representation with the parent node marked as sequential.

6.5. Activity, Workflow Model and Instance Ontology

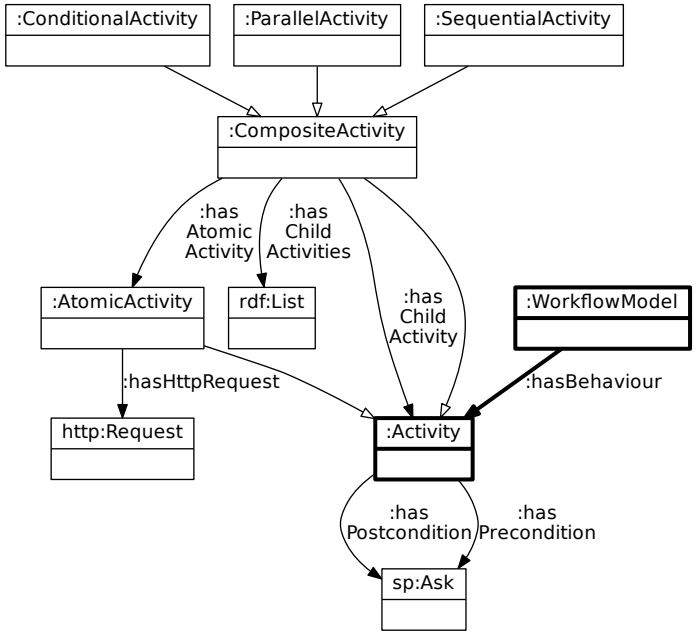
To describe workflow models and instances as well as activities, we propose an ontology. We developed the ontology, see Figure 6.2[2], with execution based on querying and reasoning under the OWA in mind. In this section, we define activities, workflows, and instances using the workflow in Figure 6.1 as example.

Activities We regard an atomic activity as a basic unit of work. We characterise an activity by a postcondition, which states what holds in the world's state after the activity has been executed. We give the postcondition as a SPARQL ASK query[3]. For the execution of an atomic activity, the activity description needs an HTTP request (cf. Figure 6.2). The workflow model in Figure 6.1 contains two activities (`<#relay1on>`, `<#relay2on>`), but omits postconditions and HTTP requests. For the example, the activities are described using the RDF graph in Figure 6.3: In this example, the postconditions are very close to the body of the requests. While in this example, the postconditions can be expected to be fulfilled in the next snapshot of the world state, this is not true in other scenarios. Imagine for instance that the request causes a valve to be opened such that water flows into a jar. In the postcondition, we could check for a certain threshold of the water level, i. e. the activity is only finished if the jar is filled to a certain extent.

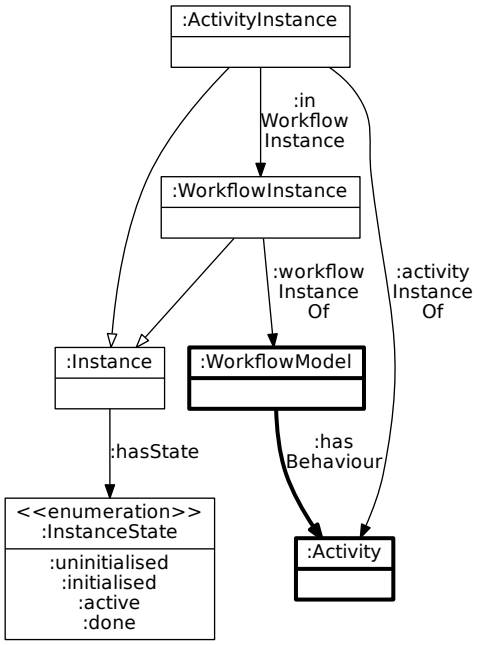
Workflow Models A workflow model is a set of activities put into a defined order. As notation to describe workflow models, BPMN is a popular choice. In the simplest form, the course of action (i. e. control flow) in a BPMN workflow model is denoted using arrows that connect activities and gateways (e. g. decisions and branches). For example 10, the middle arrow in the workflow model in Figure 6.1 orders activities `<#relay1on>` and `<#relay2on>` sequentially.

In this chapter, we assume a different representation of the control flow of a workflow model: A tree structure, as investigated by Vanhatalo et al. [215]. Tree-structured workflow languages include BPEL, a popular language to describe executable workflows. In the tree, activities are leaf nodes. The non-leaf nodes are typed, and the type determines the control flow of the children. The connection between the tree-based (dashed) and the flow-based (solid) workflow representation is depicted in Figure 6.1. Flow-based workflows can be losslessly translated to tree-structured workflows and vice versa [175]. We use the tree structure, as checks for completion of workflow parts are

[2]The ontology can be accessed at `http://purl.org/wild/vocab`
[3]`http://www.w3.org/TR/sparql11-query/`



(a) Workflow Models



(b) Workflow Instances

Figure 6.2.: The WiLD ontology to express workflow models (a) and instances (b) as UML Class Diagram. Shared classes between the diagrams are depicted in bold. We use the UML Class Diagram's class, inheritance, association, and enumeration to denote the RDFS ontology language's `rdfs:Class`, `rdfs:subClassOf`, `rdf:Property` with `rdfs:domain` and `rdfs:range`, and instances.

```
<#relay1on> a :AtomicActivity ;
  :hasHttpRequest <#req1> ;
  :hasPostcondition <#pc1> .

<#req1> a http:Request ;
  http:mthd http_m:GET ;
  http:requestURI "http://t2-relay.example/modules/relay/1"^^xsd:anyURI ;
  http:body """@prefix foaf: <http://xmlns.com/foaf/0.1/> .
               @prefix saref: <https://w3id.org/saref#> .
               <> foaf:primaryTopic <#it> .
               <#it> a saref:LightSwitch ;
                  saref:hasState saref:On .""" .

<#pc1> a sp:Ask ;
  sp:where ( [ sp:subject <http://t2-relay.example/modules/relay/1#it> ;
               sp:predicate saref:hasState ;
               sp:object saref:On ] ) .


<#relay2on> a :AtomicActivity ;
  :hasHttpRequest <#req2> ;
  :hasPostcondition <#pc2> .

<#req2> a http:Request ;
  http:mthd http_m:GET ;
  http:requestURI "http://t2-relay.example/modules/relay/2"^^xsd:anyURI ;
  http:body """@prefix foaf: <http://xmlns.com/foaf/0.1/> .
               @prefix saref: <https://w3id.org/saref#> .
               <> foaf:primaryTopic <#it> .
               <#it> a saref:LightSwitch ;
                  saref:hasState saref:On .""" .

<#pc2> a sp:Ask ;
  sp:where ( [ sp:subject <http://t2-relay.example/modules/relay/2#it> ;
               sp:predicate saref:hasState ;
               sp:object saref:On ] ) .
```

Figure 6.3.: RDF to describe the atomic activities of the example workflow model.

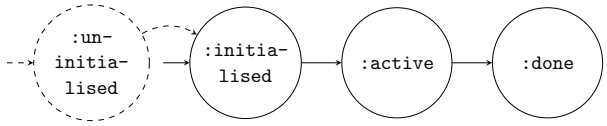


Figure 6.4.: State machine for the workflow and activity instance resources. The dashed part only concerns workflow instance resources.

easier in a tree. Of the multitude of control flow features of different workflow languages, we support the most basic and common, which have been compiled to the *basic workflow patterns* [211].

Based on the previous activity definitions in RDF, we now define the control flow of the workflow model of example 10, cf. Figure 6.1:

```
<#wfm> a :WorkflowModel ;
  :hasBehaviour <#root> .
<#root> a :SequentialActivity ;
  :hasChildActivities ( <#relay1on> <#relay2on> ) .
```

As we assume tree-structured workflows, each workflow model (`<#wfm>`) has a root activity (`<#root>`). If an activity is composite, i. e. a control flow element, then the activity has an RDF list of child activities. Here, `<#root>` is sequential, with the child activities `<#relay1on>`, `<#relay2on>`. The child activities could again be composite, thus forming a tree. Leaves in the tree (here `<#relay1on>` and `<#relay2on>`) are atomic activities. We require child activities to be given in an RDF list, which is explicitly terminated. This termination closes the set of list elements and thus allows for executing workflows under the OWA, which e. g. includes querying whether *all* child activities of a parent activity are `:done`). Yet, for the operational semantics we also need a direct connection between a parent activity and a child activity, which we derive from an RDF list using monotonic reasoning, here:

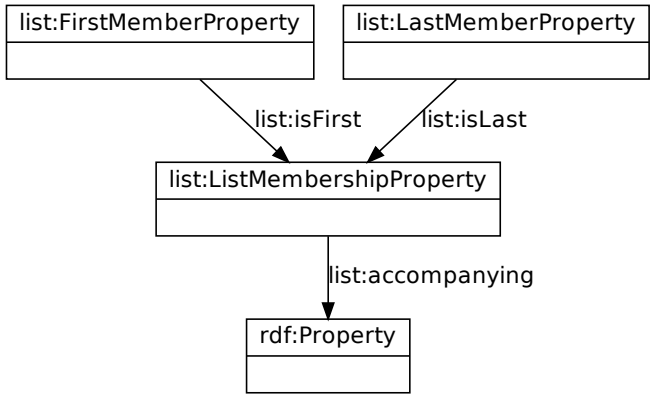
```
<#root> :hasChildActivity <#relay1on> , <#relay2on> .
```

We present details on this list processing in Section 6.6.

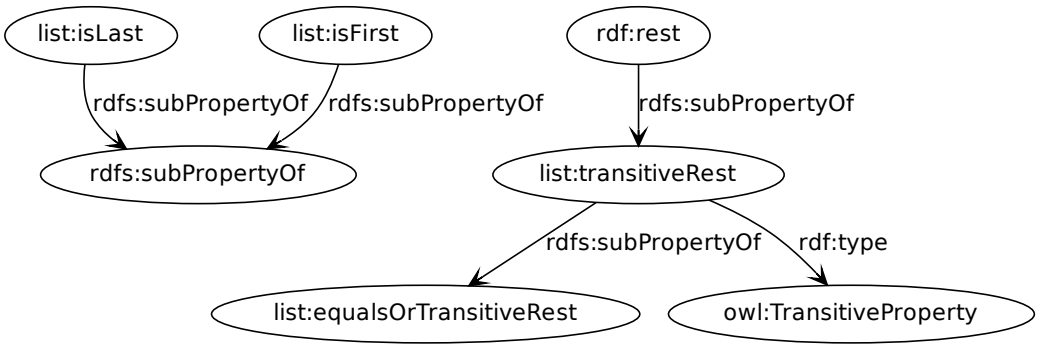
Instances Using workflow instances, we can run multiple copies of a workflow model. A workflow instance consequently consists of instances of the model's activities. We model the relation of the instances to their counterparts as shown in Figure 6.2. During and after workflow monitoring/execution, the operational semantics maintain the states of instances in an LDP container. At runtime, the instances' states evolve according to the state machine depicted in Figure 6.4 (terms from Figure 6.2). Section 6.7 is about the operationalisation of the evolution.

6.6. Reasoning and Querying over RDF Lists in OWL LD

We identified RDF lists as useful for reasoning over workflows under the open world assumption, as the RDF lists are explicitly terminated. We hence defined a vocabulary



(a) Classes and domain/range of properties as UML class diagram. UML classes denote classes in RDFS, and UML associations domain and range in RDFS.



(b) The hierarchy of properties visualised according to [47].

Figure 6.5.: Classes and properties of our ontology for RDF list processing.

and rules for list processing, which we use when we give the operational semantics of the workflow language. The vocabulary and the semantics can be found online[4] We show the classes and domain and range of the properties in Figure 6.5a and the property hierarchy in Figure 6.5b. The property `list:accompanying` defines the counterpart to a property that connects a URI to an RDF list. When describing workflows above in Section 6.5, we stated that `:hasChildActivity list:accompanying :hasChildActivity .` to connect all child activities directly to the parent activity. The semantics of the property `list:accompanying` can be given using the following rules:

```
# First element
{ ?p :accompanying ?p2 . ?s ?p2 ?list . ?list rdf:first ?listItem . }
=> { ?s ?p ?listItem . } .

# Second element
{ ?p :accompanying ?p2 . ?s ?p2 ?list . ?s ?p ?listItem .
```

[4] `http://purl.org/list/vocab` and `http://purl.org/list/semantics`, visited on 2018-06-10.

```
  ?list :transitiveRest ?bn .
  ?bn rdf:first ?listItem ; rdf:rest ?bn2 .
  ?bn2 rdf:first ?anotherListItem . }
=> { ?s ?p ?anotherListItem . } .

# >Second element
{ ?p :accompanying ?p2 . ?s ?p2 ?list . ?s ?p ?listItem .
  ?bn rdf:first ?listItem ; rdf:rest ?bn2 .
  ?bn2 rdf:first ?anotherListItem . }
=> { ?s ?p ?anotherListItem . } .
```

The properties `list:isFirst` and `list:isLast` can be used to define properties that point to the first element in a list, or the last respectively. We define the semantics of the properties as follows:

```
# First
{ ?tlmcp :isFirst ?lmp . ?lmp :accompanying ?lp .
  ?s ?lp ?list . ?list rdf:first ?leftMostChild . }
=> { ?s ?tlmcp ?leftMostChild . } .

# Last
{ ?trmcp :isLast ?lmp . ?lmp :accompanying ?lp .
  ?s ?lmp ?rightMostChild . ?s ?lp ?list .
  ?list :transitiveRest ?bn .
  ?bn rdf:first ?rightMostChild ; rdf:rest rdf:nil . }
=> { ?s ?trmcp ?rightMostChild . } .
```

6.7. Operational Semantics

In this section, we give operational semantics in rules[5] to our workflow language[6]. Before we define the rules, we give an overview of what the rules do.

6.7.1. Overview

The rules fulfil the following purposes (the numbers are only to guide the reader):

I. Retrieve state[7]

1) Retrieve the state of the writeable resources in the LDP container, which maintain the workflow/activity instances' state

[5] A corresponding Notation3 file can be found at `http://purl.org/wild/semantics`, visited on 2018-06-10.

[6] In a production environment, access control to the instances' LDP container needs to be in place to keep third parties from interfering with the monitoring/execution.

[7] A benefit of using Linked Data throughout is that we can access the workflow/activity instances' state and the world's state in a uniform manner.

2) Retrieve the relevant world state

II. Initialise workflow instances if applicable
   1) Set the root activity's instance `:active`
   2) Set the workflow instance `:initialised`
   3) Creating instance resources for all activities in the corresponding workflow model and set them `:initialised`

III. Finalise workflow instances if their root node is `:done`

IV. Execute and observe `:active` activities
   1) Execution: if an atomic activity turns `:active`, fire the HTTP request
   2) If the postcondition of an `:active` activity is fulfilled, set it `:done`

V. Advance composite activities according to control flow, which includes:
   1) Set a composite activity's children `:active`
   2) Advance between children
   3) Finalise a composite activity

6.7.2. Condition-Action Rules

We next give the rules for the listed purposes. To shorten the presentation, we factor out those rules that, for workflow execution, fire an activity's HTTP request if the activity becomes `:active`. Those rules are not needed when monitoring. The rules are of the form (the variable *method* holds the request type):

```
{ ?act a :AtomicActivity ; :hasHttpRequest ?req .
  ?req http:mthd ?method ; http:requestURI ?uri ; # etc.
} => { _:h http:mthd ?method ; http:requestURI ?uri ; # etc.
     } .
```

I. Retrieve State

The following rules specify the retrieval of data where the rule interpreter locally maintains state. Analogously, the world state can get retrieved. Either by explicitly stating URIs to be retrieved:

```
{ _:h http:mthd httpM:GET ; http:requestURI webserver:ldpcontainer . } .
```

Or by following links from data that is already known:

```
{ webserver:ldpcontainer ldp:contains ?y . }
=> { _:h http:mthd httpM:GET; http:requestURI ?y . } .
```

II. Initialise Workflow Instances

If there is an uninitialised workflow instance (e. g. injected by a third party using a POST request into the polled LDP container), the following rules create corresponding resources for the activity instances and set the workflow instance initialised:

```
{ ?wfi a :WorkflowInstance ; :hasState :uninitialised ;
    :workflowInstanceOf ?wfm . ?wfm :hasBehaviour ?act . }
=> { _:h http:mthd httpM:POST ;
       http:requestURI webserver:ldpcontainer ;
       http:body { <#it> :activityInstanceOf ?act ;
         :inWorkflowInstance ?wfi ; :hasState :active . } .
     # Also, the workflow instance is set initialised:
     _:z http:mthd httpM:PUT ;
       http:requestURI ?wfi ;
       http:body { ?wfi a :WorkflowInstance ; :hasState :initialised ;
         :workflowInstanceOf ?td . } . } .
```

Analogously, we initialise instances for the activities in the workflow model.

III. Finalise Workflow Instances

The done state of the root activity gets propagated to the workflow instance:

```
{ ?wfi a :WorkflowInstance ;
       :hasState :active ;
       :workflowInstanceOf ?wfm .
  ?wfm :hasBehaviour ?act ;
       :hasState :done . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?wfi: ;
       http:body { ?wfi a :WorkflowInstance ;
         :workflowInstanceOf ?wfm ;
         :hasState :done . } . } .
```

IV. Execute and Observe Atomic Activities

In the following, we give a rule in its entirety, which marks an activity as done if its postcondition is fulfilled.

```
{ ?wfi a :WorkflowInstance ;
       :hasState :active ;
       :workflowInstanceOf ?wfm .
  ?wfm :hasDescendantActivity ?act .
  ?act a :AtomicActivity ;
       :hasPostcondition ?postc .
  ?acti a :ActivityInstance ;
        :activityInstanceOf ?act ;
        :hasState :active .
  ?postc sparql-result:boolean true . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?acti ; http:body
```

```
     { ?acti :activityInstanceOf ?act ; :inWorkflowInstance ?wfi ;
         :hasState :done . } . } .
```

To shorten the presentation of the rules, we introduce the following simplifications: We assume that (1) we are talking about an active workflow instance, and (2) that the resource representing an instance coincides with its corresponding activity in the workflow model. (3), the PUT requests in the text do not actually overwrite the whole resource representation but patch the resources by ceteris paribus overwriting the corresponding $hasState(\cdot, \cdot)$ triple.

V. Advance According to Control Flow

In this subsection, we give the rules for advancing a workflow instance according to the basic workflow patterns [211].

Workflow Pattern 1: Sequence If there is an active sequential activity with the first activity initialised, we set this first activity to active:

```
{ ?s a :SequentialActivity ;
     :hasState :active ;
     :hasChildActivities ?child ;
  ?child rdf:first ?act1 .
  ?act1 :hasState :initialised . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?act1 ; http:body
     { ?act1 :hasState :active . } . } .
```

We advance between activities in a sequence using the following rule:

```
{ ?seq a :SequentialActivity ;
       :hasState :active ;
       :hasChildActivity ?childDone .
  ?childDone :hasState :done .
  ?childNext :hasState :initialised .
  ?list rdf:first ?childDone ;
        rdf:rest ?l .
  ?l rdf:first ?childNext . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?childNext ; http:body
     { ?childNext :hasState :active . } . } .
```

If we have reached the end of the list of children of a sequence, we regard the sequence as done (the rule is an example of the exploitation of the explicit termination of the RDF list to address the OWA):

```
{ ?seq a :SequentialActivity ;
       :hasState :active ;
       :hasChildActivity ?child .
  ?child :hasState :done .
  ?list rdf:first ?child ;
        rdf:rest rdf:nil . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?seq ; http:body
     { ?seq :hasState :done . } . } .
```

Workflow Pattern 2: Parallel Split A parallel activity consists of several activities executed simultaneously. If a parallel activity becomes active, all of its components are set to active:

```
{ ?p a :ParallelActivity ;
     :hasState :active ;
     :hasChildActivity ?child .
  ?child :hasState :initialised . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?child ; http:body
     { ?child :hasState :active . } . } .
```

Workflow Pattern 3: Synchronisation If all the components of a parallel activity are done, the whole parallel activity can be considered done. To find out whether all components of a parallel are done, we have to mark instances the following way to deal with the RDF list. First, we check whether the first element of the children of the parallel activity is done:

```
{ ?p a :ParallelActivity ;
     :hasState :active ;
     :hasChildActivities ?list .
  ?list rdf:first ?child .
  ?child :hasState :done . }
=> { ?child :hasState :doneFromListItemOne . } .
```

Then, starting from the first, we one by one check the activities in the list of child activities whether they are done.

```
{ ?p a :ParallelActivity ;
     :hasState :active ;
     :hasChildActivity ?child .
  ?child :hasState :doneFromListItemOne .
  ?childNext :hasState :done .
  ?list rdf:first ?child ;
        rdf:rest ?list2 .
   ?list2 rdf:first ?childNext . }
=> { ?childNext :hasState :doneFromListItemOne . } .
```

If the check proceeded to the last list element, the whole parallel activity is done:

```
{ ?p a :ParallelActivity ;
    :hasState :active ;
    :hasChildActivity ?child .
  ?list rdf:first ?child ;
        rdf:rest rdf:nil .
  ?child :hasState :doneFromListItemOne . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?p ; http:body
     { ?p :hasState :done . } . } .
```

Workflow Pattern 4: Exclusive Choice The control flow element choice implements a choice between different alternatives, for which conditions are specified. For the evaluation of the condition, we first have to check whether all child activities are in initialised state, similarly to the rules for Workflow Pattern 3:

```
{ ?ca a :ConditionalActivity ;
      :hasState :active ;
      :hasChildActivities ?list .
  ?list rdf:first ?child .
  ?child :hasState :initialised . }
=> { ?child :hasState :initialisedFromListItemOne . } .

{ ?ca a :ConditionalActivity ;
      :hasState :active ;
      :hasChildActivity ?child .
  ?child :hasState :initialisedFromListItemOne .
  ?list1 rdf:first ?child ;
         rdf:rest ?list2 .
  ?list2 rdf:first ?childNext .
  ?childNext :hasState :initialised . }
=> { ?childNext :hasState :initialisedFromListItemOne . } .
```

If the check succeeded, we can evaluate the conditions and set an activity active:

```
{ ?ca a :ConditionalActivity ;
      :hasState :active ;
      :hasChildActivity ?child .
  ?child :hasState :initialisedFromListItemOne ;
         :hasPrecondition ?prec .
  ?list rdf:first ?child ;
        rdf:rest rdf:nil .
  ?prec sparql-result:boolean true . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?child ; http:body
     { ?child :hasState :active . } . } .
```

We leave it to the modeller to make sure that the preconditions of the children of a conditional activity are mutually exclusive.

Workflow Pattern 5: Simple Merge If one of the children of a conditional activity is done, the whole conditional activity is done:

```
{ ?ca a :ConditionalActivity ;
     :hasState :active ;
     :hasChildActivity ?child .
  ?child :hasState :done . }
=> { _:h http:mthd httpM:PUT ; http:requestURI ?ca ; http:body
     { ?ca :hasState :done . } . } .
```

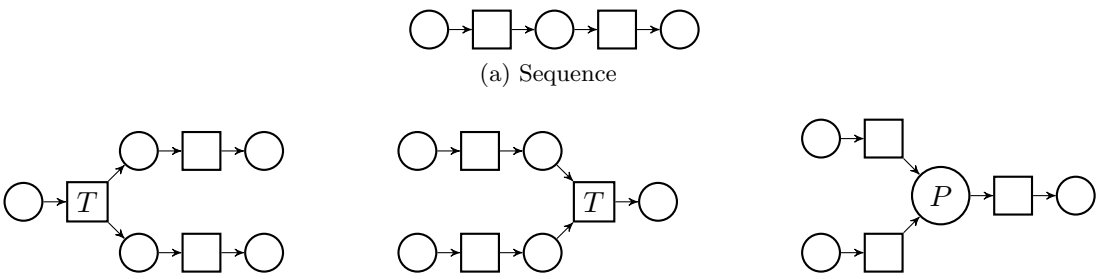
6.8. Evaluation

First, we formally show the correctness of our approach to *specifying* workflows by presenting the relationship of our operational semantics to the formal specification of the basic workflow patterns, which we support completely. Second, to show the applicability of our approach in a real-world setting, we report on how we used the approach to do *monitoring* of workflows for human-in-the-loop aircraft cockpit evaluation in Virtual Reality. Third, we empirically evaluate our approach to *executing* workflows in a Smart Building simulator.

6.8.1. Formal Evaluation

Van der Aalst et al. use Petri Nets to precisely specify the semantics of the basic workflow patterns [211]. We now show correctness by giving a mapping of our operational semantics to Petri Nets. Similar to *tokens* in a Petri Net that pass between *transitions*, our operational semantics passes the *active* state between *activities* using rules (linking to the rules for the workflow patterns from Section 6.7.2.V):

- The rule to advance between activities within a `:SequentialActivity` may only set an activity active if its preceding activity has terminated. In the Petri Net for the Sequence, a transition may only fire if the preceding transition has put a token into the preceding place, see Figure 6.6a and the workflow pattern 1 rules.
- Only after the activity before a `:ParallelActivity` has terminated, the rule to advance in a parallel activity sets all child activities active. In the Petri Net for the Parallel Split, all places following transition T get a token iff transition T has fired, see Figure 6.6b and the workflow pattern 2 rules.
- Only if all activities in a `:ParallelActivity` have terminated, the rules pass on the active state. In the Petri Net for the Synchronisation, transition T may only fire if there is a place with a token in all incoming arcs (cf. Figure 6.6c and the workflow pattern 3 rules).



(a) Sequence

(b) Parallel Split and Exclusive Choice

(c) Synchronisation

(d) Simple Merge

Figure 6.6.: Petri Nets for the Basic Workflow Patterns.

- In the `ConditionalActivity`, one child activity is chosen by the rule according to mutually exclusive conditions. Similarly, exclusive conditions determine the continuation of the flow after transition T in the Petri Net for the Exclusive Choice, see Figure 6.6b and the workflow pattern 4 rules.

- If one child activity of a `:ConditionalActivity` switches from active to done, the control flow may proceed according to the rule. Likewise, the transition following place P in the Petri Net for the Simple Merge (Figure 6.6d) may fire iff there is a token in P, cf. the workflow pattern 5 rules.

6.8.2. Applicability: The Case of Virtual Aircraft Cockpit Design

We successfully applied our approach together with industry in aircraft cockpit design [132], where workflow monitoring is used to evaluate cockpit designs regarding Standard Operating Procedures. The monitoring is traditionally done by Human Factors experts using stopwatches in physical cockpits. We built an integrated Cyber-Physical System of Virtual Reality, flight simulation, sensors, and workflows to digitise the monitoring. The challenge was to integrate the different system components on both the system interaction and the data level. We built Linked Data interfaces to the components to cover the interaction integration, and used semantic reasoning to integrate the data. Our approach allows monitoring workflows in this setting of Linked Data and semantic reasoning at runtime of the integrated system. The integrated system also contains a user interface to model workflows, which Human Factors experts evaluated as highly efficient.

6.8.3. Empirical Evaluation

We provide a performance evaluation in the same Building Automation scenario as the evaluation of Section 5.8. While in Section 5.8, we evaluated one-step automation tasks given in rules, we here use multi-step automation tasks given in workflows. In the context of the scenario of the benchmark, such tasks can include: (1) Control schemes based on time, sensor data and web data, (2) Automated supervision of cleaning personnel, (3) Presence simulation, (4) Evacuation support. Those tasks go beyond simple rule-based automation tasks as typically found in home automation (e. g. Eclipse SmartHome[8]) and on the web (e. g. IFTTT[9]), as the tasks require a notion of task instance state. We therefore model the tasks as workflow and run them as workflow instances, which access the building management systems in an integrated fashion using Read-Write Linked Data interfaces.

The environment for our benchmark is a Linked Data representation of building 3 of IBM Research Dublin. Specifically, we use the building representation 2 from the evaluation in Section 5.8. While the rule-based automation tasks concerned the automation of rooms, we here consider the workflow-based automation of whole buildings. Hence, to evaluate at different scales, we can run multiple copies of the building.

[8] `https://www.eclipse.org/smarthome/`, visited on 2018-06-10.
[9] `https://ifttt.com/`, visited on 2018-06-10.

Table 6.1.: Average runtime [s] for workflows Wn in different numbers of buildings.

	W1	W2	W3	W4	W5
1 Building	2	2	6	12	18
10 Buildings	8	9	26	61	75
20 Buildings	12	13	38	80	109
50 Buildings	19	21	61	156	218

The workload for our benchmark is the control flow of the five representative workflow models proposed by Ferme et al. [66] for evaluating workflow engines, determined by clustering workflows from literature, the web, and industry. We interpreted the five workflow models using the four automation tasks presented above: Task 1 corresponds to the first two workflow models; the subsequent tasks to the subsequent workflow models. We distinguish two types of activities in the tasks: activities that are mere checks, i. e. have only a postcondition (e. g. an hour of the day to build time-based control), and tasks that enact change (e. g. turn on a light), where we attach an HTTP request. For instance, the evacuation scenario would involve both: Checks whether certain areas of the building are empty, and corresponding signalling and locking. We assigned the types to the workflows' activities and made sure, for repeatability, that the postconditions are always fulfilled and that the requests do not interfere with the workflow.

The set-up for our evaluation using the benchmark contains a server with a 32-core Intel Xeon E5-2670 CPU and 256 GB of RAM running Debian Jessie. We use the server to run the buildings and the workflow management. For the workflow management, we deploy the operational semantics and required OWL LD reasoning on Linked Data-Fu 0.9.12[10]. We add rules for inverse properties as presented in the examples of the Brick ontology. The buildings and the workflow state are maintained in individual Eclipse Jetty servers running LDBBC 0.0.6[11] LDP implementations. We add workflow instances each 0.2 s after a warm-up time of 20 s. The workflow models and more information can be found online[12].

The results of our evaluation can be found in Table 6.1. Varying the number of activities (W1-W5), and varying the number of devices (proportional to buildings), we observe linear behaviour. The linear behaviour stems from the number of requests to be made, which depends on the number of activities and workflow instances. With no data and reasoning results reusable between buildings, there is no benefit in running the workflows for all buildings on one engine. Instead, we could run one engine per building, thus echoing the decentralisation of data.

[10] `http://linked-data-fu.github.io/`
[11] `http://github.com/kaefer3000/ldbbc`, visited on 2018-06-10.
[12] `http://people.aifb.kit.edu/co1683/2018/iswc-wild/`, visited on 2018-06-10.

6.9. Conclusion

In this chapter, we presented an approach to use workflows to specify, monitor, and execute applications that build on distributed data and functionality provided as Read-Write Linked Data. We defined a workflow ontology and corresponding operational semantics to monitor and execute workflows. We aligned our approach to the basic workflow patterns, reported on an application in Virtual Reality, and evaluated using a benchmark in an Internet of Things scenario.

The assumptions of the environment of Read-Write Linked Data (i. e. RDF and REST) present peculiar challenges for a workflow system: We work under the open-world assumption and without notifications. Our approach addresses the challenges without adding assumptions to the architecture of the environment, but by modelling a closed world where necessary and by using polling.

We believe that our approach, which brings workflows in a language that is closely related to the popular BPMN notation to Read-Write Linked Data, enables non-experts to engage in the development of applications for Read-Write Linked Data that can be verified, validated, and executed.

7. Summary and Conclusion

In this thesis, we investigated the notion of behaviour on the Linked Data web. We presented approaches for specifying, monitoring, and executing behaviour on Linked Data. We summarise by revisiting the hypotheses, research questions, challenges, and contributions, and point to directions for future research. Last, we conclude by positioning the thesis in the broader context of the Semantic Web research and application, and recent trends in industrial applications.

7.1. Specification of Behaviour

We investigated hypothesis H2 (*we can describe the dynamics of Linked Data using formal methods*) by showing on different levels that we can indeed formally describe the dynamics of Linked Data. On the lowest level, contribution C1, the Linked Data Transition System, is a descriptive approach to formally specify the dynamics of Linked Data from an omniscient observer's perspective. The Linked Data Transition System introduced the snapshot-based view on dynamic Linked Data, which we used throughout the subsequent chapters. We evaluated this view using an analysis for temporal coherence (C3). On a higher level than the Linked Data Transition System, we provided more formal accounts of dynamics Linked Data when investigating hypothesis H4 (*web technologies can serve as framework for computation and application specification*). Using contribution C4, ASM4LD, we addressed RQ3 (*how can we specify computation using Read-Write Linked Data and rules*) and provided an approach that combines the model-theoretic semantics of RDF, the semantics of HTTP messages, and rules that change state, with Abstract State Machines, a model of computation. We showed that despite the restrictions we introduced, our approach is Turing-complete. Hence, ASM4LD is a formal way to specify user agent behaviour to interact with dynamic Linked Data. Using contribution C5, WiLD, we address RQ4 (*how can we combine control flow specifications in workflows with semantic reasoning and RESTful access*) by providing an approach that combines RESTful interaction, reasoning in OWL LD, and specifications of application control flow in a workflow language that supports the basic workflow patterns. Thus, we provided three contributions to answer RQ1, *how can we describe the dynamics of Linked Data using formal methods.*

Future directions regarding the specification of behaviour on Linked Data include the investigation of suitable validation and verification techniques for our contributions C1, C4, and C5, e. g. based on Model Checking [10]. Another direction to investigate are artefact-centric workflow languages instead of the control-flow-centric WiLD, which could provide a good fit to the resource-based web architecture. Moreover, our considerations

mostly regarded user agents. Hence, another open question is the specific investigation of servers.

7.2. Monitoring of Behaviour without Specifications

To investigate hypotheses H1 (*Linked Data is dynamic across application domains*) and H3 (*we can derive coherent snapshots of Linked Data*), we built contribution C2, the Dynamic Linked Data Observatory. Building C2, we addressed the RQ2 of how to construct a corpus to study the dynamics of Linked Data. To answer the question, we extensively surveyed literature and contrasted two views on Linked Data on the web, a crawl-based and a registry-based, using an in-depth data analysis. We built our corpus on a synthesis of the two views taking into account previous undertakings to build corpora in other web-related domains and practical considerations. Based on the corpus, we investigated dynamics in Linked Data on the web on different levels: On the lowest level, we looked at the dynamics that occur when accessing Linked Data and found e. g. that documents were unavailable about 20 % of the time. We looked at the dynamics of documents and found e. g. that only 17 % of the documents never changed during our observation. Yet, we confirm H1 as we found that no application domain under consideration (i. e. topics in the classification of the LOD-cloud) consists only in static PLDs. Moreover, we looked at the dynamics of triples in documents and found that schema information is fairly static, and the most dynamics can be observed in RDF literals that contain time stamps. We conducted a statistical analysis to investigate hypothesis H3. We found that temporal coherence is fairly unlikely in snapshots of the size, the network characteristics, and the change characteristics of the Dynamic Linked Data Observatory, yet the error introduced by incoherence can be regarded as very small. For our analyses, we investigated the applicability of semantic technologies to conduct the analyses by defining an ontology for dynamic Linked Data against which we wrote analytic queries. We found that while the expressivity of SPARQL and RDF is sufficient to cover basic analyses, the performance of state of the art triple stores only partially suffices. We evaluated the Dynamic Linked Data Observatory by pointing to the uptake and the interest of the community towards the corpus.

Future directions in this area are manifold, and the community is constantly adding new analyses based on the Dynamic Linked Data Observatory (C2), mostly in the area of schema dynamics. As we are concerned with crawling to compose the Dynamic Linked Data Observatory and as ASM4LD and WiLD allow for and make use of the following of hyperlinks, it would be interesting to investigate the reliability and the repeatability of hyperlink following approaches using the Dynamic Linked Data Observatory. Another strand of future investigations is to put the results of the Dynamic Linked Data Observatory into practice, where the BEAR benchmark [69] is a good starting point, using which people can evaluate the update performance of triple stores and which uses 58 snapshots from the Dynamic Linked Data Observatory. Moreover, we believe that the application of semantic technologies in our analyses could also be a source of benchmarks. With the analytic queries regarded as continuous queries that are executed after each new snapshot

is produced, and the data regarded as snapshots of the state of networked sensors and actuators, we believe that optimising for our queries and data could help design SPARQL engines to cover monitoring use-cases, e. g. in production environments.

7.3. Execution of Behaviour and Monitoring of Behaviour with Specifications

Our contributions ASM4LD (C4) and WiLD (C5) allow not only for the specification of behaviour, but also for the execution of behaviour. In ASM4LD, we specify behaviour in rules. We can execute specifications in ASM4LD by performing rule evaluation and HTTP requests in ASM steps. Based on ASM4LD, we derived requirements for a language to specify user agents for Linked Data and gave operational semantics to the Linked Data-Fu [198] language. In WiLD, we specify behaviour using workflows. To make the specifications of WiLD executable, we provided operational semantics to the workflow language in ASM4LD. Depending on the specifications of the activities in WiLD, we can use WiLD both to monitor and to execute behaviour specified in workflow models. To evaluate ASM4LD and WiLD, we provided a benchmarking environment from the building automation domain (C6).

Future directions for research around the execution of behaviour include extensions to ASM4LD and WiLD to include the handling of errors in the context of accessing Linked Data, be it on the HTTP level or the networking level. For instance, if a user agent can successfully dereference a URI at all times, i. e. open a TCP/IP network connection, send an HTTP request and receive an HTTP response, except for one point in time, where the network connection cannot get established, what should the user agent do? When analysing dynamic Linked Data from the web in Chapter 4, we assumed that the representation received last also applies to this one point in time, in which no connection can be established. But an application given as ASM4LD, where we want to act according to the information received – should the application halt instead? An answer to this question should be formally grounded and practically relevant. Furthermore, as we consider a networked setting in which we work with data produced in a decentralised fashion, standards and practices to formally handle errors in the data are an open problem. For instance, any triple that contains a literal, which represents a numerical value greater than $2^{31} - 1$, and is typed as `xsd:int` instead of `xsd:integer`, is false under any interpretation that recognises those data types. Hence, with the logical principle *ex falso sequitur quodlibet*, we can entail any graph from graphs that contain such a triple. To address this problem, we can educate publishers [116] or apply resilient reasoning techniques [150], yet a standardised treatment is still to be developed.

7.4. Conclusion

After years of research and development, (Semantic) web technologies around Linked Data, specifically REST/HTTP and RDF, find widespread and growing mainstream adoption across industries. In this thesis, we were concerned with foundational aspects

of behaviour on Linked Data, regardless of the application domain. Using the Dynamic Linked Data Observatory, we provided empirical findings on the dynamics of Linked Data that shall help to design applications that use Linked Data and follow both established paradigms such as Triple Stores and novel paradigms such as Link Traversal-Based Query Execution. We presented ASM4LD as a rule-based model of computation for Linked Data that is Turing-complete, i. e. we presented a universal way to specify the behaviour of user agents whose state is maintained in Linked Data. Moreover, we presented WiLD to specify behaviour in workflows for monitoring and execution in the environment of Linked Data. In other words, ASM4LD and WiLD allow for composing networked applications at different levels of abstraction in an environment that has been designed for data integration. Such applications are especially relevant in the emerging application areas of the Internet of Things and Industrie 4.0, where myriads of heterogeneously described sensors and actuators from different vendors need to get composed into applications to fulfil business goals. Last, we remark from a more abstract point of view, that with ASM4LD, we provided a way to specify simple reflex agents, and with WiLD, we provided a way to specify user agents with internal state. Hence, thus specified user agents can be classified into to the lowest levels of the spectrum of intelligent agents in Artificial Intelligence [185] and therefore can serve as basis for rational agents on the web.

List of Figures

List of Tables

Acronyms

API	application programming interface.
ASM4LD	Abstract State Machines for Linked Data.
BPEL	Business Process Execution Language [3].
BPMN	Business Process Model and Notation [165].
BTC	Billion Triple Challenge.
CKAN	Comprehensive Knowledge Archive Network.
CRUD	create, retrieve, update, delete.
CWA	closed-world assumption [181].
FOAF	Friend of a Friend.
HTML	Hypertext Markup Language [112].
HTTP	Hypertext Transfer Protocol [75].
IETF	Internet Engineering Task Force.
JSON	JavaScript Object Notation [26].
LDP	Linked Data Platform [196].
LDTS	Linked Data Transition System.
LOD	Linking Open Data.
OWA	open-world assumption [181].
OWL	Web Ontology Language [113].
PLD	pay-level domain [147].
RDF	Resource Description Framework [47].
RDFS	RDF Schema [30].
REST	Representational State Transfer [73].

SIOC	Semantically-Interlinked Online Communities.
SPARQL	SPARQL Protocol and RDF Query Language [195].
URI	Uniform Resource Identifier [21].
W3C	World Wide Web Consortium.
WiLD	Workflows in Linked Data.
XML	Extensible Markup Language [27].

Bibliography

[1] Mohammad Abdel-Qader, Ansgar Scherp and Iacopo Vagliano. "Analyzing the Evolution of Vocabulary Terms and Their Impact on the LOD Cloud". In: *Proceedings of the 15th European Semantic Web Conference (ESWC)*. 2018.

[2] Keith Alexander, Richard Cyganiak, Michael Hausenblas and Jun Zhao. "Describing Linked Datasets". In: *Proceedings of the 2nd International Workshop on Linked Data on the Web (LDOW) at the 18th International Conference on World Wide Web (WWW)*. 2009.

[3] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri and Alex Yiu, eds. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. 11th Apr. 2007. URL: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (visited on 10/06/2018).

[4] James Anderson and Arto Bendiken. "Transaction-Time Queries in Dydra". In: *Joint Proceedings of the 2nd Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW) and the 3rd Workshop on Linked Data Quality (LDQ) at the 13th European Semantic Web Conference (ESWC)*. 2016.

[5] Anupriya Ankolekar, Frank Huch and Katia P. Sycara. "Concurrent Semantics for the Web Services Specification Language DAML-S". In: *Proceedings of the 5th International Conference on Coordination Models and Languages*. 2002.

[6] Carlos Buil Aranda, Aidan Hogan, Jürgen Umbrich and Pierre-Yves Vandenbussche. "SPARQL Web-Querying Infrastructure: Ready for Action?" In: *Proceedings of the 12th International Semantic Web Conference (ISWC)*. 2013.

[7] Marcelo Arenas, Sebastián Conca and Jorge Pérez. "Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard". In: *Proceedings of the 21st World Wide Web Conference (WWW)*. 2012.

[8] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak and Zachary G. Ives. "DBpedia: A Nucleus for a Web of Open Data". In: *Proceedings of the 6th International Semantic Web Conference (ISWC) and 2nd Asian Semantic Web Conference (ASWC)*. 2007.

[9] Sören Auer, Jan Demter, Michael Martin and Jens Lehmann. "LODStats - An Extensible Framework for High-Performance Dataset Analytics". In: *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*. 2012.

[10] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[11] Bharathan Balaji, Arka Aloke Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh E. Gupta, Mikkel Baun Kjærgaard, Mani B. Srivastava and Kamin Whitehouse. "Brick: Towards a Unified Metadata Schema For Buildings". In: *Proceedings of the 3rd International Conference on Systems for Energy-Efficient Built Environments (BuildSys)*. ACM. 2016.

[12] Dave J. Beckett. "The design and implementation of the redland RDF application framework". In: *Proceedings of the 10th International World Wide Web Conference (WWW)*. 2001.

[13] 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: *Proceedings of the 13th International Semantic Web Conference (ISWC)*. 2014.

[14] *Behaviour*. Oxford Living Dictionaries. URL: `https://en.oxforddictionaries.com/definition/behaviour` (visited on 01/05/2018).

[15] Tim Berners-Lee. *Information Management: A Proposal*. 1989. URL: `http://www.w3.org/History/1989/proposal.html`.

[16] Tim Berners-Lee. "WWW: Past, Present, and Future". In: *Computer* 29.10 (1996).

[17] Tim Berners-Lee. *Linked Data*. Design Issues. 2006. URL: `http://www.w3.org/DesignIssues/LinkedData.html`.

[18] Tim Berners-Lee. *Read-Write Linked Data*. Design Issues. 2009. URL: `http://www.w3.org/DesignIssues/ReadWriteLinkedData.html`.

[19] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer and David Sheets. "Tabulator: Exploring and analyzing Linked Data on the Semantic Web". In: *Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI) at the 5th International Semantic Web Conference (ISWC)*. 2006.

[20] Tim Berners-Lee and Dan Connolly. *Notation3 (N3): A readable RDF syntax*. Team Submission. W3C. Mar. 2011. URL: `http://www.w3.org/TeamSubmission/n3/`.

[21] Tim Berners-Lee, Roy Fielding and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. Internet Standard. RFC 3986. IETF, Jan. 2005. URL: `http://www.ietf.org/rfc/rfc3986.txt`.

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

[23] Gaetano Borriello and Roy Want. "Embedded Computation Meets the World Wide Web". In: *Communications of the ACM* 43.5 (2000).

[24] Paolo Bouquet, Chiara Ghidini and Luciano Serafini. “Querying the Web of Data: A Formal Approach”. In: *Proceedings of the 4th Asian Semantic Web Conference (ASWC)*. 2009.

[25] Tim Bray. “Measuring the Web”. In: *Computer Networks* 28.7-11 (1996).

[26] Tim Bray, ed. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259 (Internet Standard). IETF, June 2017. URL: `http://www.ietf.org/rfc/rfc8259.txt`.

[27] Tim Bray, Jean Paoli, Michael Sperberg-McQueen, Eve Maler, François Yergeau and John Cowan. *Extensible Markup Language (XML) 1.1 (Second Edition)*. Recommendation. W3C, Aug. 2006. URL: `http://www.w3.org/TR/xml11/`.

[28] Brian E. Brewington and George Cybenko. “How dynamic is the Web?” In: *Computer Networks* 33.1-6 (2000).

[29] Brian E. Brewington and George Cybenko. “Keeping Up with the Changing Web”. In: *IEEE Computer* 33.5 (2000).

[30] Dan Brickley and Ramanathan Guha, eds. *RDF Schema 1.1*. Recommendation. W3C, Feb. 2014. URL: `http://www.w3.org/TR/rdf-schema/`.

[31] Sergey Brin and Lawrence Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks* 30.1-7 (1998).

[32] Jeen Broekstra, Arjohn Kampman and Frank van Harmelen. “Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema”. In: *Proceedings of the 1st International Semantic Web Conference (ISWC)*. 2002.

[33] Alison Callahan, Jose Cruz-Toledo, Peter Ansell and Michel Dumontier. “Bio2RDF Release 2: Improved Coverage, Interoperability and Provenance of Life Science Linked Data”. In: *Proceedings of the 10th European Semantic Web Conference (ESWC)*. 2013.

[34] Sarven Capadisli, Amy Guy, Christoph Lange, Sören Auer, Andrei Vlad Sambra and Tim Berners-Lee. “Linked Data Notifications: A Resource-Centric Communication Protocol”. In: *Proceedings of the 14th European Semantic Web Conference (ESWC)*. 2017.

[35] Fabio Casati, Stefano Ceri, Barbara Pernici and Giuseppe Pozzi. “Deriving Active Rules for Workflow Enactment”. In: *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA)*. 1996.

[36] Ana Cerdeira-Pena, Antonio Fariña, Javier D. Fernández and Miguel A. Martínez-Prieto. “Self-Indexing RDF Archives”. In: *Proceedings of the 27th Data Compression Conference (DCC)*. 2016.

[37] Soumen Chakrabarti, Martin van den Berg and Byron Dom. “Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery”. In: *Computer Networks* 31.11-16 (1999).

[38] Senthilanand Chandrasekaran, John A. Miller, Gregory A. Silver, Ismailcem Budak Arpinar and Amit P. Sheth. "Performance Analysis and Simulation of Composite Web Services". In: *Electronic Markets* 13.2 (2003).

[39] K. Mani Chandy and Leslie Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems". In: *ACM Transactions on Computer Systems (TOCS)* 3.1 (1985).

[40] Victor Charpenay, Sebastian Käbisch and Harald Kosch. "Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things". In: *Proceedings of the 1st Workshop on SemanticWeb technologies for the Internet of Things (SWIT) at the 15th International Semantic Web Conference (ISWC)*. 2016.

[41] Junghoo Cho and Hector Garcia-Molina. "Synchronizing a Database to Improve Freshness". In: *Proceedings of the ACM 26th International Conference on Management of Data (SIGMOD)*. 2000.

[42] Junghoo Cho and Hector Garcia-Molina. "The Evolution of the Web and Implications for an Incremental Crawler". In: *Proceedings of 26th International Conference on Very Large Data Bases (VLDB)*. 2000.

[43] Junghoo Cho and Hector Garcia-Molina. "Effective page refresh policies for Web crawlers". In: *ACM Transactions on Database Systems (TODS)* 28.4 (2003).

[44] Junghoo Cho and Hector Garcia-Molina. "Estimating frequency of change". In: *ACM Transactions on Internet Technology (TOIT)* 3.3 (2003).

[45] Andrei Ciortea, Olivier Boissier, Antoine Zimmermann and Adina Magda Florea. "Responsive Decentralized Composition of Service Mashups for the Internet of Things". In: *Proceedings of the 6th International Conference on the Internet of Things (IoT)*. 2016.

[46] Edward G. Coffman Jr., Zhen Liu and Richard R. Weber. "Optimal robot scheduling for web search engines". In: *Journal of Scheduling* 1.1 (1998).

[47] Richard Cyganiak, David Wood and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Recommendation. W3C, Feb. 2014. URL: `http://www.w3.org/TR/rdf11-concepts/`.

[48] Mathieu d'Aquin and Enrico Motta. "Watson, more than a Semantic Web search engine". In: *Semantic Web Journal* 2.1 (2011).

[49] Laura Daniele, Frank T. H. den Hartog and Jasper Roes. "Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology". In: *Proceedings of the 7th International Workshop on Formal Ontologies meet Industry (FOMI)*. 2015.

[50] Frank DeRemer and Hans H. Kron. "Programming-in-the-Large Versus Programming-in-the-Small". In: *IEEE Transactions on Software Engineering* 2.2 (1976).

[51] Edsger Wybe Dijkstra. "On the role of scientific thought". In: *Selected Writings on Computing: A Personal Perspective*. Springer, Aug. 1974.

[52] Li Ding and Timothy W. Finin. “Characterizing the Semantic Web on the Web”. In: *Proceedings of the 5th International Semantic Web Conference (ISWC)*. 2006.

[53] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi and Joel Sachs. “Swoogle: a search and metadata engine for the semantic web”. In: *Proceedings of the 13th International Conference on Information and Knowledge Management (CKIM)*. ACM. 2004.

[54] Renata Queiroz Dividino, Thomas Gottron and Ansgar Scherp. “Strategies for Efficiently Keeping Local Linked Open Data Caches Up-To-Date”. In: *Proceedings of the 14th International Semantic Web Conference (ISWC)*. Vol. 2. 2015.

[55] Renata Queiroz Dividino, Thomas Gottron, Ansgar Scherp and Gerd Gröner. “From Changes to Dynamics: Dynamics Analysis of Linked Open Data Sources”. In: *Proceedings of the 1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data (PROFILES) at the 11th Extended Semantic Web Conference (ESWC)*. 2014.

[56] Renata Queiroz Dividino, André Kramer and Thomas Gottron. “An Investigation of HTTP Header Information for Detecting Changes of Linked Open Data Sources”. In: *Proceedings of Posters & Demos at the 11th European Semantic Web Conference (ESWC)*. 2014.

[57] Renata Queiroz Dividino, Ansgar Scherp, Gerd Gröner and Thomas Grotton. “Change-a-LOD: Does the Schema on the Linked Data Cloud Change or Not?” In: *Proceedings of the 4th International Workshop on Consuming Linked Data (COLD) at the 12th International Semantic Web Conference (ISWC)*. 2013.

[58] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy and Jeffrey C. Mogul. “Rate of Change and other Metrics: a Live Study of the World Wide Web”. In: *Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems (USITS)*. 1997.

[59] Donald E. Eastlake 3rd and Aliza R. Panitz. *Reserved Top Level DNS Names*. RFC 2606 (Best Current Practice). Updated by RFC 6761. Internet Engineering Task Force, June 1999. URL: `http://www.ietf.org/rfc/rfc2606.txt`.

[60] Marc Ehrig and Alexander Mädche. “Ontology-Focused Crawling of Web Documents”. In: *Proceedings of the 18th ACM Symposium on Applied Computing (SAC)*. 2003.

[61] Erik Elmroth, Francisco Hernández-Rodriguez and Johan Tordsson. “Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment”. In: *Future Generation Computer Systems* 26.2 (2010).

[62] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui and Anne-Marie Kermarrec. “The many faces of publish/subscribe”. In: *ACM Computing Surveys* 35.2 (2003).

[63] Peter C. Evans and Rahul C. Basole. “Revealing the API Ecosystem and Enterprise Strategy via Visual Analytics”. In: *Communications of the ACM* 59.2 (2016).

[64] Lee Feigenbaum, Gregory Williams, Kendall Clark and Elias Torres. *SPARQL 1.1 Protocol*. Recommendation. W3C, Mar. 2013. URL: http://www.w3.org/TR/sparql11-protocol/.

[65] Dieter Fensel. "Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information". In: *Proceedings of the International Conference on Intelligence in Communication Systems (INTELLCOMM)*. 2004.

[66] Vincenzo Ferme, Marigianna Skouradaki, Ana Ivanchikj, Cesare Pautasso and Frank Leymann. "Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions". In: *Proceedings of the 18th International Conference on Business Process Modeling, Development, and Support (BPMDS)*. 2017.

[67] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres and Mario Arias. "Binary RDF representation for publication and exchange (HDT)". In: *Journal of Web Semantics* 19 (2013).

[68] Javier D. Fernández, Miguel A. Martínez-Prieto, Axel Polleres and Julian Reindorf. "HDTQ: Managing RDF Datasets in Compressed Space". In: *Proceedings of the 15th European Semantic Web Conference (ESWC)*. 2018.

[69] Javier D. Fernández, Jürgen Umbrich and Axel Polleres. *BEAR: Benchmarking the Efficiency of RDF Archiving*. Tech. rep. 02/2015. Department of Information Systems and Operations, WU Vienna, Austria, 2015.

[70] Javier D. Fernández, Jürgen Umbrich, Axel Polleres and Magnus Knuth. "Evaluating Query and Storage Strategies for RDF Archives". In: *Proceedings of the 12th International Conference on Semantic Systems (SEMANTICS)*. 2016.

[71] I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455 (Proposed Standard). Internet Engineering Task Force, Dec. 2011. URL: http://www.ietf.org/rfc/rfc6455.txt.

[72] Dennis Fetterly, Mark S. Manasse, Marc Najork and Janet L. Wiener. "A large-scale study of the evolution of web pages". In: *Proceedings of the 12th International Conference on World Wide Web (WWW)*. 2003.

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

[74] Roy Fielding and Julian Reschke, eds. *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*. RFC 7232 (Proposed Standard). IETF, June 2014. URL: http://www.ietf.org/rfc/rfc7232.txt.

[75] Roy Fielding and Julian Reschke, eds. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230 (Proposed Standard). IETF, June 2014. URL: http://www.ietf.org/rfc/rfc7230.txt.

[76] Roy Fielding and Julian Reschke, eds. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231 (Proposed Standard). IETF, June 2014. URL: http://www.ietf.org/rfc/rfc7231.txt.

[77] Valeria Fionda, Giuseppe Pirrò and Claudio Gutiérrez. "NautiLOD: A Formal Language for the Web of Data Graph". In: *Transactions on the Web* 9.1 (2015).

[78] Martin Fowler. *Richardson Maturity Model*. Mar. 2010. URL: `http://martinfowler.com/articles/richardsonMaturityModel.html` (visited on 12/12/2016).

[79] Michael P. Gallaher, Alan C. O'Connor, John L. Dettbarn Jr. and Linda T. Gilday. *Cost analysis of inadequate interoperability in the US capital facilities industry*. NIST GCR 04-867. 2004.

[80] David Garlan, Robert Allen and John Ockerbloom. "Architectural Mismatch: Why Reuse Is So Hard". In: *IEEE Software* 12.6 (1995).

[81] Paula Gearon, Alexandre Passant and Axel Polleres, eds. *SPARQL 1.1 Update*. Recommendation. W3C, Mar. 2013. URL: `http://www.w3.org/TR/sparql11-update/`.

[82] David Gelernter. "Generative Communication in Linda". In: *Transactions on Programming Languages and Systems (TOPLAS)* 7.1 (1985). ACM.

[83] Yolanda Gil and Varun Ratnakar. "A Comparison of (Semantic) Markup Languages". In: *Proceedings of the 15th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*. 2002.

[84] Yolanda Gil, Varun Ratnakar, Ewa Deelman, Gaurang Mehta and Jihie Kim. "Wings for Pegasus". In: *Proceedings of the 19th Conference on Innovative Applications of Artificial Intelligence (IAAI)*. 2007.

[85] Birte Glimm, Aidan Hogan, Markus Krötzsch and Axel Polleres. "OWL: Yet to arrive on the Web of Data?" In: *Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW)*. 2012.

[86] Thomas Gottron. "Of Sampling and Smoothing: Approximating Distributions over Linked Open Data". In: *Proceedings of the 1st International Workshop on Dataset PROFIling & fEderated Search for Linked Data (PROFILES) at the 11th Extended Semantic Web Conference (ESWC)*. 2014.

[87] Thomas Gottron and Christian Gottron. "Perplexity of Index Models over Evolving Linked Data". In: *Proceedings of the 11th European Semantic Web Conference (ESWC)*. 2014.

[88] Carrie Grimes and Sean O'Brien. "Microscale evolution of web pages". In: *Proceedings of Posters at the 17th International Conference on World Wide Web (WWW)*. 2008.

[89] Christophe Guéret, Paul T. Groth, Frank van Harmelen and Stefan Schlobach. "Finding the Achilles Heel of the Web of Data: Using Network Analysis for Link-Recommendation". In: *Proceedings of the 9th International Semantic Web Conference (ISWC)*. 2010.

[90] Yuri Gurevich. "Evolving Algebras 1993: Lipari Guide". In: *Specification and Validation Methods*. Ed. by Egon Börger. Oxford University Press, 1995.

[91] Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren and Christoph Bussler. "WSMX - A Semantic Service-Oriented Architecture". In: *Proceedings of the 3rd International Conference on Web Services (ICWS)*. 2005.

[92] Armin Haller, Krzysztof Janowicz, Simon Cox, Danh Le Phuoc, Kerry Taylor and Maxime Lefrançois. *Semantic Sensor Network Ontology*. Recommendation. W3C, Oct. 2017. URL: `https://www.w3.org/TR/vocab-ssn/`.

[93] Steven Harris and Andy Seaborne, eds. *SPARQL 1.1 Query Language*. Recommendation. W3C, Mar. 2013. URL: `http://www.w3.org/TR/sparql11-query/`.

[94] Andreas Harth. *Billion Triples Challenge data set*. 2009. URL: `http://km.aifb.kit.edu/projects/btc-2009/` (visited on 10/06/2018).

[95] Andreas Harth. *Billion Triples Challenge data set*. 2010. URL: `http://km.aifb.kit.edu/projects/btc-2010/` (visited on 10/06/2018).

[96] Andreas Harth. *Billion Triples Challenge data set*. 2011. URL: `http://km.aifb.kit.edu/projects/btc-2011/` (visited on 10/06/2018).

[97] Andreas Harth. *Billion Triples Challenge data set*. 2012. URL: `http://km.aifb.kit.edu/projects/btc-2012/` (visited on 10/06/2018).

[98] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler and Jürgen Umbrich. "Data summaries for on-demand queries over Linked Data". In: *Proceedings of the 19th International Conference on World Wide Web (WWW)*. 2010.

[99] Andreas Harth and Tobias Käfer. "Towards Specification and Execution of Linked Systems". In: *Proceedings of the 28th Workshop Grundlagen von Datenbanken (GvD)*. GI. 2016.

[100] Andreas Harth and Tobias Käfer. "Specifying and Executing Application Behaviour with Condition-Request Rules". In: *Proceedings of the Workshop on Decentralizing the Semantic Web at the 16th International Semantic Web Conference (ISWC)*. 2017.

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

[102] Andreas Harth, Jürgen Umbrich and Stefan Decker. "MultiCrawler: A Pipelined Architecture for Crawling and Indexing Semantic Web Data". In: *Proceedings of the 5th International Semantic Web Conference (ISWC)*. 2006.

[103] Olaf Hartig. "How Caching Improves Efficiency and Result Completeness for Querying Linked Data". In: *Proceedings of the 4th International Workshop on Linked Data on the Web (LDOW) at the 20th International Conference on World-Wide Web (WWW)*. 2011.

[104] Olaf Hartig. "Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution". In: *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*. 2011.

[105] Olaf Hartig, Christian Bizer and Johann Christoph Freytag. "Executing SPARQL Queries over the Web of Linked Data". In: *Proceedings of the 8th International Semantic Web Conference (ISWC)*. 2009.

[106] Olaf Hartig and Jorge Pérez. "LDQL: A query language for the Web of Linked Data". In: *Web Semantics* 41 (2016).

[107] Bernhard Haslhofer and Erich J. Neuhold. "A Retrospective on Semantics and Interoperability Research". In: *Foundations for the Web of Information and Services – A Review of 20 Years of Semantic Web Research*. Springer, 2011.

[108] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab and Aidan Hogan. "SPORTAL: Profiling the Content of Public SPARQL Endpoints". In: *International Journal on Semantic Web and Information Systems* 12.3 (2016).

[109] Patrick Hayes and Peter Patel-Schneider, eds. *RDF 1.1 Semantics*. Recommendation. W3C, Feb. 2014. URL: `http://www.w3.org/TR/rdf11-mt/`.

[110] James A. Hendler. "Where Are All the Intelligent Agents?" In: *IEEE Intelligent Systems* 22.3 (2007).

[111] Antonio Garrote Hernández and María N. Moreno García. "A formal definition of RESTful semantic web services". In: *Proceedings of the First International Workshop on RESTful Design (WS-REST)*. 2010.

[112] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor and Silvia Pfeiffer. *HTML5*. Recommendation. W3C, Oct. 2014. URL: `http://www.w3.org/TR/html5/`.

[113] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter Patel-Schneider and Sebastian Rudolph. *OWL 2 Web Ontology Language Primer (Second Edition)*. Recommendation. W3C, Dec. 2012. URL: `http://www.w3.org/TR/owl2-primer/`.

[114] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph and York Sure. *Semantic Web*. Heidelberg, Germany: Springer, 2008.

[115] Aidan Hogan. "Skolemising Blank Nodes while Preserving Isomorphism". In: *Proceedings of the 24th International Conference on World Wide Web (WWW)*. 2015.

[116] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker and Axel Polleres. "Weaving the Pedantic Web". In: *Proceedings of the 3rd International Workshop on Linked Data on the Web (LDOW) at the 19th International Conference on World Wide Web (WWW)*. 2010.

[117] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres and Stefan Decker. "Searching and browsing Linked Data with SWSE: The Semantic Web Search Engine". In: *Journal of Web Semantics* 9.4 (2011).

[118] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[119] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Member Submission. W3C, May 2004. URL: `http://www.w3.org/Submission/SWRL/`.

[120] Arnaud Le Hors, Martin Nally and Steve Speicher. "Using Read/Write Linked Data for Application Integration – Towards a Linked Data Basic Profile". In: *Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW)*. 2012.

[121] Zhisheng Huang and Heiner Stuckenschmidt. "Reasoning with Multi-version Ontologies: A Temporal Logic Approach". In: *Proceedings of the 4th International Semantic Web Conference (ISWC)*. 2005.

[122] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno F. Terry Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya and Roman Vaculín. "Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles". In: *Proceedings of the 7th International Workshop on Web Services and Formal Methods (WS-FM)*. 2011.

[123] Kingsley Uyi Idehen. *Semantic Web Layer Cake Tweak, Explained*. 13th July 2017. URL: `https://medium.com/openlink-software-blog/semantic-web-layer-cake-tweak-explained-6ba5c6ac3fab` (visited on 10/06/2018).

[124] Robert Isele, Jürgen Umbrich, Christian Bizer and Andreas Harth. "LDspider: An Open-source Crawling Framework for the Web of Linked Data". In: *Proceedings of the Posters & Demonstration at the 9th International Semantic Web Conference (ISWC)*. 2010.

[125] Stefan Jablonski and Christoph Bussler. *Workflow management – modeling concepts, architecture and implementation*. International Thomson, 1996.

[126] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One*. Recommendation. W3C, Dec. 2004. URL: `http://www.w3.org/TR/webarch/`.

[127] Herbert Jaeger. "Dynamical Systems – A Navigational Guide". Tutorial at the 18th Interdisciplinary College (IK). Günne am Möhnesee, Germany, 2016.

[128] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne and Aidan Hogan. "Observing Linked Data Dynamics". In: *Proceedings of the 10th European Semantic Web Conference (ESWC)*. 2013.

[129] Tobias Käfer and Andreas Harth. *Billion Triples Challenge data set*. 2014. URL: `http://km.aifb.kit.edu/projects/btc-2014/` (visited on 10/06/2018).

[130] Tobias Käfer and Andreas Harth. "Rule-based Programming of User Agents for Linked Data". In: *Proceedings of the 11th International Workshop on Linked Data on the Web (LDOW) at the Web Conference (27th WWW)*. 2018.

[131] Tobias Käfer and Andreas Harth. "Specifying, Monitoring, and Executing Workflows in Linked Data Environments". In: *Proceedings of the 17th International Semantic Web Conference (ISWC)*. 2018.

[132] Tobias Käfer, Andreas Harth and Sébastien Mamessier. “Towards declarative programming and querying in a distributed Cyber-Physical System: The i-VISION case”. In: *Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPSData) at the 9th CPS week*. 2016.

[133] Tobias Käfer, Sebastian Lauber and Andreas Harth. “Using Workflows to Build Compositions of Read-Write Linked Data APIs on the Web of Things”. In: *Proceedings of Posters & Demos at the 17th International Semantic Web Conference (ISWC)*. 2018.

[134] Tobias Käfer, Jürgen Umbrich, Aidan Hogan and Axel Polleres. “Towards a Dynamic Linked Data Observatory”. In: *Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW)*. 2012.

[135] Tobias Käfer, Alexandra Wins and Maribel Acosta. “Modelling and Analysing Dynamic Linked Data using RDF and SPARQL”. In: *Proceedings of the 4th International Workshop on Dataset PROFILing and fEderated Search for Web Data (PROFILES) at the 16th International Semantic Web Conference (ISWC)*. 2017. Best Paper.

[136] Yiping Ke, Lin Deng, Wilfred Ng and Dik Lun Lee. “Web dynamics and their ramifications for the development of Web search engines”. In: *Computer Networks* 50.10 (2006).

[137] Felix Leif Keppmann, Tobias Käfer, Steffen Stadtmüller, René Schubotz and Andreas Harth. “High Performance Linked Data Processing for Virtual Reality Environments”. In: *Proceedings of Posters & Demos at the 13th International Semantic Web Conference (ISWC)*. 2014.

[138] Michael Kifer. “Deductive and Object Data Languages: A Quest for Integration”. In: *Proceedings of the 4th International Conference on Deductive and Object-Oriented Databases*. 1995.

[139] Tim Kindberg, John J. Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra and Mirjana Spasojevic. “People, places, things: Web presence for the real world”. In: *Proceedings of the 3rd Workshop on Mobile Computing Systems and Applications (WMCSA)*. IEEE. 2000.

[140] Michel C. A. Klein and Dieter Fensel. “Ontology versioning on the Semantic Web”. In: *Proceedings of the 1st Semantic Web Working Symposium (SWWS)*. 2001.

[141] Michel C. A. Klein, Dieter Fensel, Atanas Kiryakov and Damyan Ognyanov. “Ontology Versioning and Change Detection on the Web”. In: *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*. 2002.

[142] Wallace Koehler. “An Analysis of Web Page and Web Site Constancy and Permanence”. In: *Journal of the American Society for Information Science and Technology* 50.2 (1999).

[143] Wallace Koehler. "Web page change and persistence - A four-year longitudinal study". In: *Journal of the American Society for Information Science and Technology* 53.2 (2002).

[144] Martijn Koster. "Robots in the Web: Threat or Treat?" In: *ConneXions – The Interoperability Report* 9 (4 Apr. 1995).

[145] Robert Kowalski. *In a comment to his blog post "The Sad State Concerning the Relationships between Logic, Rules and Logic Programming" replying to a question by Andreas Harth.* 2nd Feb. 2015. URL: `http://disq.us/p/u9od8e`.

[146] Markus Lanthaler and Christian Gütl. "Hydra: A Vocabulary for Hypermedia-Driven Web APIs". In: *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW) at the 26th International Conference on World Wide Web (WWW).* 2013.

[147] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang and Dmitri Loguinov. "IRLbot: scaling to 6 billion pages and beyond". In: *Proceedings of the 17th International Conference on World Wide Web, (WWW).* 2008.

[148] Martin Leinberger, Ralf Lämmel and Steffen Staab. "The Essence of Functional Programming on Semantic Data". In: *Proceedings of the 26th European Symposium on Programming (ESOP) at the European Joint Conferences on Theory and Practice of Software (ETAPS).* 2017.

[149] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter and Ramesh C. Agarwal. "Characterizing Web Document Change". In: *Proceedings of the 2nd International Conference on Advances in Web-Age Information Management (WAIM).* 2001.

[150] Yue Ma and Pascal Hitzler. "Paraconsistent Reasoning for OWL 2". In: *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems (RR).* 2009.

[151] Matoula Magiridou, S. Sahtouris, Vassilis Christophides and Manolis Koubarakis. "RUL: A Declarative Update Language for RDF". In: *Proceedings of the 4th International Semantic Web Conference (ISWC).* 2005.

[152] James Martin. *Managing the Data-base Environment.* Englewood Cliffs, NJ, USA: Prentice-Hall, 1983.

[153] Brian McBride. "Jena: Implementing the RDF Model and Syntax Specification". In: *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb).* 2001.

[154] Albert Meroño-Peñuela, Peter Wittek and Sándor Darányi. "Visualizing the Drift of Linked Open Data Using Self-Organizing Maps". In: *Proceedings of the 1st Workshop on Detection, Representation and Management of Concept Drift in Linked Open Data (Drift-a-LOD) at the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW).* 2017.

[155] Robin Milner, Joachim Parrow and David Walker. "A Calculus of Mobile Processes, I". In: *Journal of Information and Computation* 100.1 (1992).

[156] Robin Milner, Joachim Parrow and David Walker. "A Calculus of Mobile Processes, II". In: *Journal of Information and Computation* 100.1 (1992).

[157] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu and Achille Fokoue. *OWL 2 Web Ontology Language Profiles (Second Edition)*. Recommendation. W3C, Dec. 2012. URL: `http://www.w3.org/TR/owl2-profiles/`.

[158] Srini Narayanan and Sheila A. McIlraith. "Simulation, verification and automated composition of web services". In: *Proceedings of the 11th International Conference on World Wide Web (WWW)*. 2002.

[159] Chifumi Nishioka and Ansgar Scherp. "Temporal Patterns and Periodicity of Entity Dynamics in the Linked Open Data Cloud". In: *Proceedings of the 8th International Conference on Knowledge Capture (K-CAP)*. 2015.

[160] Chifumi Nishioka and Ansgar Scherp. "Information-theoretic Analysis of Entity Dynamics on the Linked Open Data Cloud". In: *Proceedings of the 3rd International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES '16) co-located with the 13th ESWC 2016 Conference, Anissaras, Greece, May 30, 2016.* 2016.

[161] Chifumi Nishioka and Ansgar Scherp. "Keeping linked open data caches up-to-date by predicting the life-time of RDF triples". In: *Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, August 23-26, 2017*. 2017.

[162] Benjamin Nowack. *The Semantic Web Technology Stack (not a piece of cake...)* 8th July 2009. URL: `http://bnode.org/media/2009/07/08/semantic_web_technology_stack.png`.

[163] Alexandros Ntoulas, Junghoo Cho and Christopher Olston. "What's new on the web?: the evolution of the web from a search engine perspective". In: *Proceedings of the 13th International Conference on World Wide Web (WWW)*. 2004.

[164] Marilena Oita and Pierre Senellart. "Deriving Dynamics of Web Pages: A Survey". In: *Proceedings of the 4th International Workshop on Linked Data on the Web (LDOW) at the 20th International Conference on World-Wide Web (WWW)*. 2011.

[165] OMG, ed. *Business Process Model And Notation – Version 2.0*. Specification. 3rd Jan. 2011. URL: `http://www.omg.org/spec/BPMN/2.0`.

[166] Kevin R. Page, David De Roure and Kirk Martinez. "REST and Linked Data: a match made for domain driven development?" In: *Proceedings of the 2nd International Workshop on RESTful Design (WS-REST)*. 2011.

[167] Sandeep Pandey, Krithi Ramamritham and Soumen Chakrabarti. "Monitoring the dynamic web to respond to continuous queries". In: *Proceedings of the 12th International Conference on World Wide Web (WWW)*. 2003.

[168] Alexandre Passant and Pablo N. Mendes. "sparqlPuSH: Proactive Notification of Data Updates in RDF Stores Using PubSubHubbub". In: *Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web (SFSW) at the 7th Extended Semantic Web Conference (ESWC)*. 2011.

[169] Cesare Pautasso. "BPEL for REST". In: *Proceedings of the 6th International Conference on Business Process Management (BPM)*. 2008.

[170] Cesare Pautasso. "RESTful Web Service Composition with BPEL for REST". In: *Data and Knowledge Engineering* 68.9 (2009).

[171] Cesare Pautasso, Ana Ivanchikj and Silvia Schreier. "A pattern language for RESTful conversations". In: *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP)*. 2016.

[172] Cesare Pautasso and Erik Wilde. "Push-Enabling RESTful Business Processes". In: *Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC)*. 2011.

[173] Cesare Pautasso and Olaf Zimmermann. "The Web as a Software Connector". In: *IEEE Software* 35.1 (2018).

[174] Cesare Pautasso, Olaf Zimmermann and Frank Leymann. "RESTful Web Services vs. 'Big' Web Services". In: *Proceedings of the 17th International Conference on World Wide Web, (WWW)*. 2008.

[175] Artem Polyvyanyy, Luciano García-Bañuelos and Marlon Dumas. "Structuring Acyclic Process Models". In: *Proceedings of the 8th International Conference on Business Process Management (BPM)*. 2010.

[176] Niko Popitsch and Bernhard Haslhofer. "DSNotify: handling broken links in the web of data". In: *Proceedings of the 19th International Conference on World Wide Web (WWW)*. 2010.

[177] Jon Postel, ed. *Internet Protocol*. RFC 791 (Internet Standard). IETF, Sept. 1981. URL: `http://www.ietf.org/rfc/rfc791.txt`.

[178] Jon Postel, ed. *Transmission Control Protocol*. RFC 793 (Internet Standard). IETF, Sept. 1981. URL: `http://www.ietf.org/rfc/rfc793.txt`.

[179] Eric Prud'hommeaux and Carlos Buil Aranda, eds. *SPARQL 1.1 Federated Query*. Recommendation. W3C, Mar. 2013. URL: `http://www.w3.org/TR/sparql11-federated-query/`.

[180] Eric Prud'hommeaux and Gavin Carothers, eds. *RDF 1.1 Turtle*. Recommendation. W3C, Feb. 2014. URL: `http://www.w3.org/TR/turtle/`.

[181] Raymond Reiter. "On Closed World Data Bases". In: *Proceedings of the Symposium on Logic and Data Bases*. 1978.

[182] Eric Rescorla, ed. *HTTP Over TLS*. RFC 2818 (Informational). IETF, June 2017. URL: `http://www.ietf.org/rfc/rfc2818.txt`.

[183] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler and Dieter Fensel. "Web Service Modeling Ontology". In: *Applied Ontology* 1.1 (2005).

[184] Marco Rospocher, Chiara Ghidini and Luciano Serafini. "An ontology for the Business Process Modelling Notation". In: *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS)*. 2014.

[185] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995.

[186] Leo Sauermann and Richard Cyganiak, eds. *Cool URIs for the Semantic Web*. SWEO IG Note. W3C, 2008. URL: `http://www.w3.org/TR/cooluris/`.

[187] Johann Schaible, Thomas Gottron and Ansgar Scherp. "TermPicker: Enabling the Reuse of Vocabulary Terms by Exploiting Data from the Linked Open Data Cloud". In: *Proceedings of the 13th European Semantic Web Conference (ESWC)*. 2016.

[188] Claudia Schon and Steffen Staab. "Towards SPARQL Instance-Level Update in the Presence of OWL-DL TBoxes". In: *Proceedings of the Joint Ontology Workshops (JOWO) Episode 3: The Tyrolean Autumn of Ontology*. 2018.

[189] Nigel Shadbolt, Tim Berners-Lee and Wendy Hall. "The Semantic Web Revisited". In: *IEEE Intelligent Systems* 21.3 (2006).

[190] Claude Elwood Shannon. "A Mathematical Theory of Communication". In: *The Bell Systems Technical Journal* 27 (3 July 1948).

[191] Amit P. Sheth. "Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics". In: *Interoperating Geographic Information Systems*. Boston, MA, USA: Springer, 1999.

[192] Joshua Shinavier. "Ripple: Functional Programs as Linked Data". In: *Proceedings of the Workshop on Scripting for the Semantic Web (SFSW) at the 4th European Semantic Web Conference (ESWC)*. 2007.

[193] Michael Sintek and Stefan Decker. "TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web". In: *Proceedings of the 1st International Semantic Web Conference (ISWC)*. 2002.

[194] Marc Spaniol, Dimitar Denev, Arturas Mazeika, Gerhard Weikum and Pierre Senellart. "Data quality in web archiving". In: *Proceedings of the 3rd Workshop on Information Credibility on the Web (WICOW) at the 18th International Conference on World Wide Web (WWW)*. 2009.

[195] *SPARQL 1.1 Overview*. Recommendation. W3C, Mar. 2013. URL: `http://www.w3.org/TR/sparql11-overview/`.

[196] Steve Speicher, John Arwe and Ashok Malhotra, eds. *Linked Data Platform 1.0*. Recommendation. W3C, Feb. 2015. URL: `http://www.w3.org/TR/ldp/`.

[197] Steffen Stadtmüller. "Dynamic Interaction and Manipulation of Web Resources". PhD thesis. Karlsruhe Institute of Technology, 2015. URL: `http://nbn-resolving.de/urn:nbn:de:swb:90-519359`.

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

[199] Vlad Stirbu. "Towards a RESTful Plug and Play Experience in the Web of Things". In: *Proceedings of the 2th IEEE International Conference on Semantic Computing (ICSC)*. 2008.

[200] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben and Jeen Broekstra. "Index structures and algorithms for querying distributed RDF repositories". In: *Proceedings of the 13th International Conference on World Wide Web (WWW)*. 2004.

[201] Rudi Studer, V. Richard Benjamins and Dieter Fensel. "Knowledge Engineering: Principles and Methods". In: *Data and Knowledge Engineering* 25.1-2 (1998).

[202] Ruben Taelman, Miel Vander Sande, Ruben Verborgh and Erik Mannens. "Versioned Triple Pattern Fragments: A Low-cost Linked Data Interface Feature for Web Archives". In: *Joint proceedings of the 3rd Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW) and the 4th Workshop on Linked Data Quality (LDQ) at the 14th European Semantic Web Conference (ESWC)*. 2017.

[203] Andreas Tolk, Saikou Y. Diallo and Jose J. Padilla. "Semiotics, entropy, and interoperability of simulation systems: mathematical foundations of M&S standardization". In: *Proceedings of the 44th Winter Simulation Conference (WSC)*. 2012.

[204] Giovanni Tummarello, Renaud Delbru and Eyal Oren. "Sindice.com: Weaving the Open Linked Data". In: *Proceedings of the 6th International Semantic Web Conference (ISWC) and 2nd Asian Semantic Web Conference (ASWC)*. 2007.

[205] Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres and Stefan Decker. "Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources". In: *Proceedings of the 3rd International Workshop on Linked Data on the Web (LDOW) at the 19th International Conference on World Wide Web (WWW)*. 2010.

[206] Jürgen Umbrich, Aidan Hogan, Axel Polleres and Stefan Decker. "Link traversal querying for a diverse Web of Data". In: *Semantic Web* 6.6 (2015).

[207] Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan and Josiane Xavier Parreira. "Hybrid SPARQL Queries: Fresh vs. Fast Results". In: *Proceedings of the 11th International Semantic Web Conference (ISWC)*. 2012.

[208] Jürgen Umbrich, Boris Villazón-Terrazas and Michael Hausenblas. "Dataset Dynamics Compendium: A Comparative Study". In: *Proceedings of the 1st International Workshop on Consuming Linked Data (COLD) at the 9th International Semantic Web Conference (ISWC)*. 2010.

[209] UNEP and SKANSKA. *Energy Efficiency in Buildings – Guidance for Facility Managers*. 2009. URL: https://group.skanska.com/48dc86/globalassets/sustainability/environmental-responsibility/energy/unep_energy-effic-broch-final.pdf (visited on 10/06/2018).

[210] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze and Michael L. Littman. "Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes". In: *Proceedings of the 34th Conference on Human Factors in Computing Systems (CHI)*. 2016.

[211] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski and Alistair P. Barros. "Workflow Patterns". In: *Distributed and Parallel Databases* 14.1 (2003).

[212] Wil M. P. van der Aalst, Mathias Weske and Dolf Grünbauer. "Case handling: a new paradigm for business process support". In: *Data and Knowledge Engineering* 53.2 (2005).

[213] Pascal van Eck, Joeri Engelfriet, Dieter Fensel, Frank van Harmelen, Yde Venema and Mark Willems. "A Survey of Languages for Specifying Dynamics: A Knowledge Engineering Perspective". In: *Transactions on Knowledge and Data Engineering (TKDE)* 13.3 (2001).

[214] Max van Kleek, Brennan Moore, David R. Karger, Paul André and m. c. schraefel. "Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web". In: *Proceedings of the 19th International Conference on World Wide Web (WWW)*. 2010.

[215] Jussi Vanhatalo, Hagen Völzer and Jana Koehler. "The Refined Process Structure Tree". In: *Proceedings of the 6th International Conference on Business Process Management (BPM)*. 2008.

[216] 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 (2016).

[217] Ruben Verborgh, Thomas Steiner, Davy van Deursen, Sam Coppens, Joaquim Gabarró Vallés and Rik van de Walle. "Functional descriptions as the bridge between hypermedia APIs and the Semantic Web". In: *Proceedings of the 3rd International Workshop on RESTful Design (WS-REST) at the 25th International Conference on World Wide Web (WWW)*. 2012.

[218] Julius Volz, Christian Bizer, Martin Gaedke and Georgi Kobilarov. "Discovering and Maintaining Links on the Web of Data". In: *Proceedings of the 8th International Semantic Web Conference (ISWC)*. 2009.

[219] W3C OWL WG. *OWL 2 Web Ontology Language Document Overview*. Recommendation. W3C, Oct. 2009. URL: http://www.w3.org/TR/owl2-overview/.

[220] Erik Wilde. *Putting Things to REST*. Tech. rep. 2007-015. iSchool, University of Berkeley, CA, USA, Nov. 2007. URL: http://dret.net/netdret/docs/wilde-irep07-015-restful-things.pdf (visited on 07/06/2018).

[221] Erik Wilde. *REST and RDF Granularity*. May 2009. URL: http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html (visited on 10/06/2018).

[222] Lina Yao, Quan Z. Sheng and Schahram Dustdar. "Web-Based Management of the Internet of Things". In: *IEEE Internet Computing* 19.4 (2015).

[223] Dogan Yazar and Adam Dunkels. "Efficient Application Integration in IP-based Sensor Networks". In: *Proceedings of the First Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys) at the 7th Conference on Embedded Network Sensor Systems (SenSys)*. ACM. 2009.

[224] Jin Yu, Boualem Benatallah, Fabio Casati and Florian Daniel. "Understanding Mashup Development". In: *IEEE Internet Computing* 12.5 (2008).

[225] Antoine Zimmermann, ed. *RDF 1.1: On Semantics of RDF Datasets*. RDF WG Note. W3C, Feb. 2014. URL: http://www.w3.org/TR/rdf11-datasets/.

[226] Michael zur Muehlen, Jeffrey V. Nickerson and Keith D. Swenson. "Developing Web Services Choreography Standards". In: *Decision Support Systems* 40.1 (2005).

A. Queries

A.1. Preprocessing

A.1.1. Semantics of Unavailable Sources

```
# Run repeatedly until no new data is added.

PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

# Fill an unsuccessful observation with the body of the previous observation.
INSERT {
  ?respNext http:body ?body .
}
WHERE {
  ?obs
    :hasSeedURI ?uri ;
    :hasSnapshot ?snap ;
    :hasLastResponse ?resp .

  ?snap
    :next ?snapNext .

  ?obsNext
    :hasSeedURI ?uri ;
    :hasSnapshot ?snapNext ;
    :hasLastResponse ?respNext .

    ?resp
    http:body ?body .

  FILTER NOT EXISTS {
      ?respNext http:body ?bodyNext .
  }
};

# Fill an unsuccessful observation with the body of the subsequent observation,
# if the unsuccessful observation does not have a successful predecessor.
INSERT {
  ?respRightBefore http:body ?body1 .
}
WHERE {
```

```
  ?obs1
    :hasSeedURI ?uri ;
    :hasSnapshot ?snap1 ;
    :hasLastResponse ?resp1 .

  ?resp1
    http:body ?body1 .

  ?snapRightBefore
    :next ?snap1 .

  ?obsRightBefore
    :hasSeedURI ?uri ;
    :hasSnapshot ?snapRightBefore ;
    :hasLastResponse ?respRightBefore .

  FILTER NOT EXISTS {
    ?obsAnyBefore
      :hasSeedURI ?uri ;
      :hasSnapshot ?snapAnyBefore .

    ?snapAnyBefore
      :next* ?snapRightBefore .

    ?obsAnyBefore
      :hasLastResponse ?respAnyBefore .
    ?respAnyBefore
      http:body ?bodyAnyBefore .
  }
}
```

A.1.2. Materialising Intermediary Results

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX crypto: <http://www.w3.org/2000/10/swap/crypto#>

INSERT { ?observation2 :changed ?changed }
WHERE {
    ?snapshot1 :next ?snapshot2 .

    ?observation2
      :hasSeedURI ?seedURI ;
      :hasSnapshot ?snapshot2 ;
      :hasLastResponse ?resp2 .
    ?resp2 http:body ?body2 .

    ?observation1
      :hasSeedURI ?seedURI ;
```

```
      :hasSnapshot ?snapshot1 ;
      :hasLastResponse ?resp1 .
    ?resp1 http:body ?body1
    ?body1 crypto:md5 ?hash1 .

    BIND(NOT EXISTS{?body2 crypto:md5 ?hash1}
      as ?changed)
}
```

A.2. High-Level Statistics

A.2.1. Number of PLDs Whose URIs Ever Dereferenced

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT (COUNT(DISTINCT ?pld) AS ?pldCount)
WHERE {
  ?pld :hosts ?uri .
  ?obs :hasSeedURI ?uri ;
    :hasLastResponse ?resp .
  ?resp http:statusCodeValue 200 .
}
```

A.2.2. Number of PLDs Whose URIs Dereferenced Per Snapshot

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT ?snapshot (COUNT(DISTINCT ?pld) AS ?pldCount)
WHERE {
  ?pld :hosts ?uri .
  ?obs :hasSeedURI ?uri ;
    :hasSnapshot ?snap ;
    :hasLastResponse ?resp .
  ?resp http:statusCodeValue 200 .
} GROUP BY ?snapshot
```

A.2.3. Number of URIs That Ever Dereferenced

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT (COUNT(DISTINCT ?uri) AS ?uriCount)
WHERE {
  ?obs :hasSeedURI ?uri ;
    :hasLastResponse ?resp .
  ?resp http:statusCodeValue 200 .
}
```

A.2.4. Number of URIs That Dereferenced Per Snapshot

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT ?snapshot (COUNT(DISTINCT ?uri) AS ?uriCount)
WHERE {
  ?obs :hasSeedURI ?uri ;
    :hasSnapshot ?snap ;
    :hasLastResponse ?resp .
  ?resp http:statusCodeValue 200 .
} GROUP BY ?snapshot
```

A.3. Queries to Analyse Dynamic Linked Data

A.3.1. Appearance

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT (COUNT(?seedURI) AS ?uris) ?NoSnapshots WHERE {
  {
  SELECT (COUNT(?snapshot1) AS ?NoSnapshots) ?seedURI  WHERE {
      ?observation :hasSnapshot ?snapshot1;
                   :hasSeedURI ?seedURI;
                   :hasLastResponse ?res .
      ?res http:statusCodeValue "200"^^xsd:integer .
    } GROUP BY ?seedURI
  }
} GROUP BY ?NoSnapshots
```

A.3.2. HTTP Responses

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT ?snapshot
       (CONCAT(?scDigit1, "xx") AS ?statusClass)
       (COUNT(DISTINCT ?seedURI) AS ?seedUriCount)
WHERE { ?observation :hasSnapshot ?snapUri ;
                     :hasSeedURI ?seedURI ;
                     :hasLastResponse ?res .
        ?res http:statusCodeValue ?sc .
        ?snapUri dc:created ?snapshot .
        BIND(SUBSTR(STR(?sc), 1, 1) AS ?scDigit1)
} GROUP BY ?scDigit1 ?snapshot
```

A.3.3. Death Rate I

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT (COUNT(?seedURI) AS ?numberOfURIcountsWith200PresentOrFuture)
       ?snapshotFromWhichToDetermineHistoryForward
WHERE {
  ?observation
    :hasSeedURI ?seedURI ;
    :hasSnapshot ?snapshotFromWhichToDetermineHistoryForward .

  ?snapshotFromWhichToDetermineHistoryForwardMinusOne
    :next ?snapshotFromWhichToDetermineHistoryForward .

  FILTER(EXISTS {
    ?snapshotFromWhichToDetermineHistoryForwardMinusOne
      :next* ?snap2 .

    ?observation2 :hasSeedURI ?seedURI ;
      :hasSnapshot ?snap2 .
      :hasLastResponse ?res2 .
      http:statusCodeValue 200 .
    }
  )
} GROUP BY ?snapshotFromWhichToDetermineHistoryForward
```

A.3.4. Death Rate II

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT (COUNT(?seedURI) AS ?numberOfURIcountsWithNon404PresentOrFuture)
       ?snapshotFromWhichToDetermineHistoryForward
WHERE {
  ?observation :hasSeedURI ?seedURI ;
    :hasSnapshot ?snapshotFromWhichToDetermineHistoryForward .

  ?snapshotFromWhichToDetermineHistoryForwardMinusOne
    :next ?snapshotFromWhichToDetermineHistoryForward .

  FILTER(EXISTS {
    ?snapshotFromWhichToDetermineHistoryForwardMinusOne :next* ?snap2 .
    ?observation2 :hasSeedURI ?seedURI ;
      :hasSnapshot ?snap2 ;
      :hasLastResponse ?res2 .
    ?res2 http:statusCodeValue ?scv .
    FILTER(?scv != "404"^^xsd:integer)
  })
} GROUP BY ?snapshotFromWhichToDetermineHistoryForward
```

A.3.5. Changes and Change Frequency

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX crypto: <http://www.w3.org/2000/10/swap/crypto#>

SELECT ?numberOfChanges (COUNT (?numberOfChanges) as ?numberOfUris)
WHERE {
  SELECT ?seedURI (COUNT(?seedURI) AS ?numberOfChanges)
  WHERE {
    ?observation2
      :hasSeedURI ?seedURI ;
      :hasSnapshot ?snapshot2 ;
      :hasLastResponse ?resp2 .
    ?resp2 http:body ?body2 .

    ?snapshot1 :next ?snapshot2 .

    ?observation1
      :hasSeedURI ?seedURI ;
      :hasSnapshot ?snapshot1 ;
      :hasLastResponse ?resp1.
    ?resp1 http:body ?body1 .
    ?body1 crypto:md5 ?hash1 .

    FILTER(NOT EXISTS{
      ?body2 crypto:md5 ?hash1 .
    })
  } GROUP BY ?seedURI
} GROUP BY ?numberOfChanges
```

A.3.6. Change Frequency and Change Amount

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX crypto: <http://www.w3.org/2000/10/swap/crypto#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?pld
       (IF(SUM(?changedAtAll) = 0, 0, SUM(?numberOfChanges) / SUM(?changedAtAll))
        AS ?meanNoChangesInChangingDocs)
       (SUM(?changedAtAll) / COUNT(?seedURI) AS ?ratioOfChangingDocs)
       (COUNT (?seedURI) AS ?noSeedURIs)
WHERE {
  SELECT ?seedURI
         (SUM(?changed) AS ?numberOfChanges)
         (IF(SUM(?changed) > 0, 1, 0) AS ?changedAtAll)
  WHERE {
    ?observation2
      :hasSeedURI ?seedURI ;
```

```
      :hasSnapshot ?snapshot2 ;
      :hasLastResponse ?resp2 .
    ?resp2 http:body ?body2 .

    ?snapshot1 :next ?snapshot2 .

    BIND(
      IF(
        EXISTS {
          ?observation1
            :hasSeedURI ?seedURI ;
            :hasSnapshot ?snapshot1 ;
            :hasLastResponse ?resp1 .
          ?resp1 http:body ?body1 .
          ?body1 crypto:md5 ?hash1 .

          ?body2 crypto:md5 ?hash1 .
       }, 0, 1)
      AS ?changed)
  } GROUP BY ?seedURI

  ?plduri :hosts ?seedURI ;
    rdfs:label ?pld .
} GROUP BY ?pld
```

A.3.7. Change Frequency and Change Amount by LOD-Cloud Classification

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX crypto: <http://www.w3.org/2000/10/swap/crypto#>
PREFIX http: <http://www.w3.org/2011/http#>

SELECT ?topic (COUNT(?pld) AS ?pldcount)
       (SUM(?doccount) AS ?docccount)
       (SUM(?static) AS ?staticcount)
       (SUM(?dual) AS ?dualcount)
       (SUM(?bulk) AS ?bulkcount)
       (SUM(?active) AS ?activecount)
WHERE {
  {
    SELECT (COUNT(?snapshot2) / 2 AS ?halfSnapCount) WHERE {
      ?snapshot1 :next ?snapshot2 .
    }
  }

  {
    SELECT ?pld (COUNT(?doc) AS ?doccount) WHERE {
      ?pld :hosts ?doc .
    } GROUP BY ?pld
  }
```

```
?pld :lod2012topicClassification ?topic .

BIND(IF(?ratioOfChangingDocs > 0.5
  && ?meanNoChangesInChangingDocs > ?halfSnapCount, 1, 0) AS ?active)
BIND(IF(?ratioOfChangingDocs < 0.5
  && ?meanNoChangesInChangingDocs > ?halfSnapCount, 1, 0) AS ?dual)
BIND(IF(?ratioOfChangingDocs < 0.5
  && ?meanNoChangesInChangingDocs < ?halfSnapCount, 1, 0) AS ?static)
BIND(IF(?ratioOfChangingDocs > 0.5
  && ?meanNoChangesInChangingDocs < ?halfSnapCount, 1, 0) AS ?bulk)

{
  SELECT ?pld
       (IF(SUM(?changedAtAll) = 0,
           0,
           SUM(?numberOfChanges) / SUM(?changedAtAll))
        AS ?meanNoChangesInChangingDocs)
       (SUM(?changedAtAll) / COUNT(?seedURI)
        AS ?ratioOfChangingDocs)
  WHERE {
    {
      SELECT ?seedURI
        (SUM(?changed) AS ?numberOfChanges)
        (IF(SUM(?changed) > 0, 1, 0) AS ?changedAtAll)
      WHERE {

        ?observation2
          :hasSeedURI ?seedURI ;
          :hasSnapshot ?snapshot2 ;
          :hasLastResponse ?resp2 .
        ?resp2 http:body ?body2 .

        ?snapshot1 :next ?snapshot2 .

        BIND(
          IF(
            EXISTS {
              ?observation1
                :hasSeedURI ?seedURI ;
                :hasSnapshot ?snapshot1 ;
                :hasLastResponse ?resp1 .
              ?resp1 http:body ?body1 .

              ?body1 crypto:md5 ?hash1 .
              ?body2 crypto:md5 ?hash1 .
            }, 0, 1)
          AS ?changed)
      } GROUP BY ?seedURI
```

```
      }

    ?pld :hosts ?seedURI .

    } GROUP BY ?pld
  }
} GROUP BY ?topic
```

B. Linked Data-Fu Rules to Implement a Turing Machine

Assume the following prefix definitions:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix tm:  <http://www.example.org/turing-machine#> .
@prefix http: <http://www.w3.org/2011/http#>.
@prefix http_m: <http://www.w3.org/2011/http-methods#>.
```

Displacement to the Left

```
{ ?tm tm:hasInitialPosition ?initPos ;
      tm:hasCurrentPosition ?currPos ;
      tm:hasCurrentState    ?currState .

  ?currPos rdf:first ?currTapeSymbol ;
    rdf:rest ?currNext .
  ?nextPos rdf:rest ?currPos .

  ?tr tm:hasCurrentTapeSymbol ?currTapeSymbol ;
      tm:hasCurrentState ?currState ;
      tm:hasNextState ?nextState ;
      tm:hasDisplacement tm:L ;
      tm:hasWriteSymbol ?writeSymbol . }
=>
{ [] http:mthd http_m:PUT ;
    http:requestURI ?currPos ;
    http:body
      { ?currPos rdf:first ?writeSymbol ;
          rdf:rest ?currNext . } . } .
{ ?tm tm:hasInitialPosition ?initPos ;
      tm:hasCurrentPosition ?currPos ;
      tm:hasCurrentState    ?currState .

  ?currPos rdf:first ?currTapeSymbol ;
    rdf:rest ?currNext .
  ?nextPos rdf:rest ?currPos .
```

```
  ?tr tm:hasCurrentTapeSymbol ?currTapeSymbol ;
      tm:hasCurrentState ?currState ;
      tm:hasNextState ?nextState ;
      tm:hasDisplacement tm:L ;
      tm:hasWriteSymbol ?writeSymbol . }
=>
{ [] http:mthd http_m:PUT ;
    http:requestURI ?tm ;
    http:body
      { ?tm tm:hasInitialPosition ?initPos ;
          tm:hasCurrentPosition ?nextPos ;
          tm:hasCurrentState    ?nextState . } . } .
```

Displacement to the Right

```
{ ?tm tm:hasInitialPosition ?initPos ;
      tm:hasCurrentPosition ?currPos ;
      tm:hasCurrentState    ?currState .

  ?currPos rdf:first ?currTapeSymbol ;
    rdf:rest ?currNext .
  ?currPos rdf:rest ?nextPos .

  ?tr tm:hasCurrentTapeSymbol ?currTapeSymbol ;
      tm:hasCurrentState ?currState ;
      tm:hasNextState ?nextState ;
      tm:hasDisplacement tm:R ;
      tm:hasWriteSymbol ?writeSymbol . }
=>
{ [] http:mthd http_m:PUT ;
    http:requestURI ?currPos ;
    http:body
      { ?currPos rdf:first ?writeSymbol ;
          rdf:rest ?currNext . } . } .
{ ?tm tm:hasInitialPosition ?initPos ;
      tm:hasCurrentPosition ?currPos ;
      tm:hasCurrentState    ?currState .

  ?currPos rdf:first ?currTapeSymbol ;
    rdf:rest ?currNext .
  ?currPos rdf:rest ?nextPos .
```

```
  ?tr tm:hasCurrentTapeSymbol ?currTapeSymbol ;
      tm:hasCurrentState ?currState ;
      tm:hasNextState ?nextState ;
      tm:hasDisplacement tm:R ;
      tm:hasWriteSymbol ?writeSymbol . }
=>
{ [] http:mthd http_m:PUT ;
    http:requestURI ?tm ;
    http:body
      { ?tm tm:hasInitialPosition ?initPos ;
          tm:hasCurrentPosition ?nextPos ;
tm:hasCurrentState ?nextState . } . } .
```