Chapter 3

Architecture of Linked Data Applications

Benjamin Heitmann

Digital Enterprise Research Institute, National University of Ireland, Galway

Richard Cyganiak

Digital Enterprise Research Institute, National University of Ireland, Galway

Conor Hayes

Digital Enterprise Research Institute, National University of Ireland, Galway

Stefan Decker

Digital Enterprise Research Institute, National University of Ireland, Galway

3.1 Introduction

Before the emergence of RDF and Linked Data, Web applications have been designed around relational database standards of data representation and service. A move to an RDF-based data representation introduces challenges for the application developer in rethinking the Web application outside the standards and processes of database-driven Web development. These include, but are not limited to, the graph-based data model of RDF [171], the Linked Data principles [83], and formal and domain specific semantics [96].

To date, investigating the challenges which arise from moving to RDF-based data representation has not been given the same priority as the research on potential benefits. We argue that the lack of emphasis on simplifying the development and deployment of Linked Data and RDF-based applications has been an obstacle for real-world adoption of Semantic Web technologies and the emerging Web of Data. However, it is difficult to evaluate the adoption of Linked Data and Semantic Web technologies without empirical evidence.

Towards this goal, we present the results of a survey of more than 100 RDF-based applications, as well as a component-based, conceptual architecture for Linked Data applications which is based on this survey. We also use the survey results to identify the main implementation challenges of moving from database-driven applications to RDF-based applications. Finally, we suggest future approaches to facilitate the standardization of components and the development of software engineering tools to increase the uptake of Linked Data.

In this chapter, we first perform an *empirical survey* of RDF-based applications over most of the past decade, from 2003 to 2009. As the Linked Data principles where introduced in 2006, this allows us to describe the current state-of-the-art for developing and deploying applications using RDF and Linked Data. It also allows us to determine how far the adoption of signature research topics, such as data reuse, data integration, and reasoning has progressed.

We then use the results of this survey as the empirical foundation for a component-based *conceptual architecture* for Linked Data applications. Our conceptual architecture describes the high level components that are most often observed among the surveyed applications. These components implement the functionality that differentiates applications using RDF from database-driven applications. For each component we describe the most common implementation strategies that are suited to specific application goals and deployment scenarios. The existence of such an architecture can be seen as evidence of a convergence to a common set of principles to solve recurring development challenges arising from the use of RDF and Linked Data.

Based on the empirical survey and the conceptual architecture, we discuss the main *implementation challenges* facing developers using RDF and Linked Data. We provide an *example analysis* of an application to show how our architecture and the implementation challenges apply to a real application.

In addition we suggest *future approaches* in using the conceptual architecture to facilitate the standardization of components and the development of software engineering tools to simplify the development of Linked Data applications.

The chapter presents this perspective through the following sections: Section 3.2 presents the findings of the empirical survey, including our research method. We also discuss potential counter-arguments to the validity of our results. Based on this empirical foundation, section 3.3 proposes a conceptual architecture for Linked Data applications, and lists all components of this architecture. We also list related chapters in this book for each component as well as other related literature. We discuss the main implementation challenges which arise from using RDF and Linked Data in section 3.4. A representative example of the analysis of an application is presented in 3.5. We present several approaches to ease the development of Linked Data applications in the future in section 3.6. Finally, section 3.7 discusses related work, and section 3.8 concludes the chapter and presents future work.

3.2 An Empirical Survey of RDF-based Applications

The evolution of Linked Data and Semantic Web technologies is characterized by the constant introduction of new ideas, standards and technologies, and thus appears to be in a permanent state of flux. However, more then a decade has passed since Tim Berners-Lee et al. published their comprehensive vision for a Semantic Web [85] in 2001. This allows us to look back on the empirical evidence left behind by RDF-based applications that have been developed and deployed during this time.

Sjoberg et. al [495] argue that there are relatively few empirical studies in software engineering research, and that the priority for evaluating new technologies is lower than that of developing new technologies. We can transfer this observation to research on the engineering of RDF-based and Linked Data applications. Without empirical evidence it is difficult to evaluate the adoption rate of Linked Data and Semantic Web technologies. To provide such insights, we collect evidence from a representative sample of the population of developers deploying applications using RDF or Linked Data. In particular we perform an empirical survey of over a hundred applications from two demonstration challenges that received much attention from the research community.

The results of our survey enable us to determine the state of adoption of key research goals such as data reuse, data integration, and reasoning. In this section, we first explain our research method. Then the findings of our empirical survey are presented, after which we discuss threats to the validity of the survey.

3.2.1 Research Method of the Survey

The empirical evidence for our survey is provided by 124 applications that were submitted to two key demonstration challenges in the Semantic Web domain: (1) the "Semantic Web challenge"[1] [332] in the years 2003 to 2009 with 101 applications, which is organized as part of the International Semantic Web Conference; and (2) the "Scripting for the Semantic Web challenge"[2] in the years 2006 to 2009 with 23 applications, which is organized as part of the European Semantic Web Conference. As the Semantic Web was an emerging research topic, the majority of applicants were from research centers or university departments, though there were some industrial participants.

Each challenge awards prizes for the top entries, as selected by the judges of the respective contest. In order to apply, the developers of an application were required to provide the judges with access to a working instance of their application. In addition, a scientific paper describing the goals and the implementation of the application was also required.

[1] `http://challenge.semanticweb.org/submissions.html`
[2] `http://www.semanticscripting.org`

The applications which were submitted to the "Semantic Web challenge" had to fulfill a set of minimal requirements, as described in [332]: The information sources of the application had to be geographically distributed, should have had diverse ownership so that there was no control of the evolution of the data; the data was required to be heterogeneous and from a real-world use-case. The applications should have assumed an open world model, meaning that the information never was complete. Finally, the applications were required to use a formal description of the data's meaning. Applications submitted to the "Scripting for the Semantic Web challenge" only had to be implemented using a scripting programming language.

The applications from the two challenges are from very different domains, such as life sciences, cultural heritage, geo-information systems, as well as media monitoring, remote collaboration, and general application development support. The Bio2RDF [416] project has converted 30 life sciences datasets into RDF and then generated links between the data sets in a consistent way. DBpedia Mobile [74] is a location-aware client for mobile phones which displays nearby locations taken from DBpedia. As a related project, LinkedGeoData [54] transforms data from the OpenStreetMap project into RDF and interlinks it with other spatial data sets. Then there is the Semantic MediaWiki [346] platform, which extends the MediaWiki platform used by Wikipedia with semantic capabilities, such as templates for entities and consistency checks. On the more applied side of the spectrum, there is MediaWatch [468], which crawls news articles for concepts related to climate change, extracts ontology concepts and provides a unified navigation interface. The MultimediaN E-Culture demonstrator [476] enables searching and browsing of cultural heritage catalogs of multiple museums in the Netherlands. As a final example, NASA has developed the SemanticOrganizer [328] which supports the remote collaboration of distributed and multidisciplinary teams of scientists, engineers, and accident investigators with Semantic technologies.

We collected the data about each application through a questionnaire that covered the details about the way in which each application implemented Semantic Web technologies and standards. It consisted of 12 questions that covered the following areas: (1) usage of programming languages, RDF libraries, Semantic Web standards, schemas, vocabularies, and ontologies; (2) data reuse capabilities for data import, export, or authoring; (3) implementation of data integration and schema alignment; (4) use of inferencing; (5) data access capabilities for the usage of decentralized sources, usage of data with multiple owners, data with heterogeneous formats, data updates, and adherence to the Linked Data principles. The full data of the survey and a description of all the questions and possible answers is available online[3].

For each application the data was collected in two steps: First, the application details were filled into the questionnaire based on our own analysis of the paper submitted with the application. Then we contacted the authors of each

[3]`http://the-blank.net/semwebappsurvey/`

paper and asked them to verify or correct the data about their application. This allowed us to fill in questions about aspects of an application that might not have been covered on a paper. For instance the implementation details of the data integration are not discussed by most papers. 65% of authors replied to this request for validation.

3.2.2 Findings

The results of our empirical survey show that the adoption of the capabilities that characterize Semantic Web technologies has steadily increased over the course of the demonstration challenges. In this section we present the results of the survey and analyze the most salient trends in the data. Tables 3.1, 3.2 and 3.3 summarize the main findings.

Table 3.1 shows the survey results about the programming languages and RDF libraries that were used to implement the surveyed applications. In addition, it shows the most implemented Semantic Web standards and the most supported schemas, vocabularies, and ontologies. Java was the most popular programming language choice throughout the survey time-span. This accords with the fact that the two most mature and popular RDF libraries, Jena and Sesame, both require Java. On the side of the scripting languages, the most mature RDF library is ARC, which explains the popularity of PHP for scripting Semantic Web applications. From 2006 there is a noticeable consolidation trend towards supporting RDF, OWL, and SPARQL, reflecting an emergent community consensus. The survey data about vocabulary support requires a different interpretation: While in 2009 the support for the top three vocabularies went down, the total number of supported vocabularies went from 9 in 2003 to 19 in 2009. This can be explained by the diversity of domains for which Linked Data became available.

The simplification of data integration is claimed as one of the central benefits of implementing Semantic Web technologies. Table 3.2 shows the implementation of data integration grouped by implementation strategy and year. The results suggest that, for a majority of applications, data integration still requires manual inspection of the data and human creation of rules, scripts, and other means of integration. However the number of applications that implement fully automatic integration is on the rise, while the number of applications that require manual integration of the data is steadily declining. The steady number of applications that do not require data integration can be explained by the availability of homogeneous sources of RDF and Linked Data that do not need to be integrated.

Table 3.3 shows the survey results on the support for particular features over the time period. Some capabilities have been steadily supported by a majority of applications throughout the whole period, while other capabilities have seen increases recently or have suffered decreases.

Support for decentralized sources, data from multiple owners, and sources with heterogeneous formats is supported each year by a large majority of the

TABLE 3.1: Implementation details by year, top 3 entries per cell

	2003	2004	2005	2006
Programming languages	Java 60% C 20%	Java 56% JS 12%	Java 66%	Java 10% JS 15% PHP 26%
RDF libraries	—	Jena 18% Sesame 12% Lucene 18%	—	RAP 15% RDFLib 10%
SemWeb standards	RDF 100% OWL 30%	RDF 87% RDFS 37% OWL 37%	RDF 66% OWL 66% RDFS 50%	RDF 89% OWL 42% SPARQL 15%
Schemas/ vocabularies/ ontologies	RSS 20% FOAF 20% DC 20%	DC 12% SWRC 12%	—	FOAF 26% RSS 15% Bibtex 10%
	2007	2008	2009	overall
Programming languages	Java 50% PHP 25%	Java 43% PHP 21%	Java 46% JS 23% PHP 23%	Java 48% PHP 19% JS 13%
RDF libraries	Sesame 33% Jena 8%	Sesame 17% ARC 17% Jena 13%	Sesame 23%	Sesame 19% Jena 9%
SemWeb standards	RDF 100% SPARQL 50% OWL 41%	RDF 100% SPARQL 17% OWL 10%	RDF 100% SPARQL 69% OWL 46%	RDF 96% OWL 43% SPARQL 41%
Schemas/ vocabularies/ ontologies	FOAF 41% DC 20% SIOC 20%	FOAF 30% DC 21% DBpedia 13%	FOAF 34% DC 15% SKOS 15%	FOAF 27% DC 13% SIOC 7%

applications. We can attribute this to the central role that these capabilities have in all of the early foundational standards of the Semantic Web such as RDF and OWL. While support for importing and exporting data has fluctuated over the years, it still remains a popular feature. Since 2007, the SPARQL standard has been increasingly used to implement APIs for importing data.

The Linked Data principles are supported by more than half of the contest entries in 2009, after being formulated in 2006 by Tim Berners-Lee [83]. Interestingly the increasing usage of the Linking Open Data (LOD) cloud may explain some of the positive and negative trends we observe in other capabilities. For example, the negative correlation (-0.61) between support for linked data principles and inferencing is probably explained by the fact that data from the LOD cloud already uses RDF and usually does not require any integration.

Support for the creation of new structured data is strongly correlated (0.75) with support for the Linked Data principles. This can be explained by the growing maturity of RDF stores and their APIs for creating data, as well as

TABLE 3.2: Data integration by implementation strategy and year

	2003	2004	2005	2006	2007	2008	2009
manual	30%	13%	0%	16%	9%	5%	4%
semi-automatic	70%	31%	100%	47%	58%	65%	61%
automatic	0%	25%	0%	11%	13%	4%	19%
not needed	0%	31%	0%	26%	20%	26%	16%

TABLE 3.3: Results of the binary properties from the application questionnaire

	2003	2004	2005	2006	2007	2008	2009
Data creation	20%	37%	50%	52%	37%	52%	76%
Data import	70%	50%	83%	52%	70%	86%	73%
Data export	70%	56%	83%	68%	79%	86%	73%
Inferencing	60%	68%	83%	57%	79%	52%	42%
Decentralized sources	90%	75%	100%	57%	41%	95%	96%
Multiple owners	90%	93%	100%	89%	83%	91%	88%
Heterogeneous formats	90%	87%	100%	89%	87%	78%	88%
Data updates	90%	75%	83%	78%	45%	73%	50%
Linked Data principles	0%	0%	0%	5%	25%	26%	65%

increased community support for adding to the Linked Data cloud. On the other hand, support for updating data after it was acquired is strongly negatively correlated (-0.74) with the growth in support for the LOD principles. This could reflect a tendency to treat structured data as being static.

To summarize: Support for features which differentiate RDF-based from database-driven applications is now widespread among the development communities of the challenges. Support for the graph-based data model of RDF is virtually at 100%, and most applications reuse formal domain semantics such as the FOAF, DublinCore, and SIOC vocabularies. Support for the Linked Data principles has reached more then half of the contest entries in 2009. And while the choice of implemented standards has crystallized around RDF, OWL, and SPARQL, the choice of programming languages and RDF libraries has gravitated towards Java, Sesame, and Jena, as well as PHP and ARC. In addition, the majority of applications still require some human intervention for the integration service.

3.2.3 Discussion of Threats to Validity

We will discuss and deflect the three most commonly raised threats to the validity of our empirical survey. These are (i) the representativeness, (ii) the number of authors validating the analysis of their applications and (iii) the reliance on analyzing academic papers instead of source code.

The first threat to validity concerns the selection of applications for our empirical survey. We choose to use only applications from two academic demonstration challenges for our survey. This may raise the question of representativeness, as most of the applications are academic prototypes. However, as both challenges explicitly also invited and received some industry submissions, the two challenges do not have a purely academic audience.

In addition, several of the applications from the "Semantic Web challenge" were successfully commercialized in the form of start-ups: The Sindice [423] crawler and lookup index for Linked Data resources became the foundation for SindiceTech.com which provides products in the area of knowledge data clouds. The Seevl application [432], a challenge winner in the year 2011, provides music recommendations and additional context for musical artists on YouTube and became a successful start-up. The Event Media [329] application aggregates information about media events and interlinks them to Linked Data. Event Media was a winner of the challenge in 2012, and entered a partnership with Nokia.

Furthermore, we believe that the data of our survey is representative for the following reasons: First, the applications could not just be tools or libraries, but had to be complete applications that demonstrate the benefits of RDF or Linked Data to the end-user of the application. Second, the applications submitted to the challenges already follow baseline criteria as required in [332], e.g. most of them use real-world data. Third, the authors of each submitted application were also required to submit a scientific paper focusing on the implementation aspects of their application, which allows them to explain the goals of their application in their own terms. Finally, each challenge was held over multiple years, which makes it easier to compare the submissions from the different years in order to see trends, such as the uptake of the Linked Data principles.

The second threat to validity is the number of authors who verified the data about their applications. Only 65% of authors replied to this request for validation. This may be due to the short-lived nature of academic email addresses. In several instances, we tried to find an alternative email address, if an address had expired. Every author that we successfully contacted validated the data about their application, which usually included only small corrections for one or two properties. We were unable to validate the data for authors that could not be reached. As the first challenge was already held in 2003, we do not believe that a much higher validation rate would have been possible at the time of writing.

The third threat to validity is that for each application only the associated academic paper and not the source code were analyzed for the survey. There are several reasons for this. At the time of writing most demonstration prototypes, except those from the most recent years, were not deployed anymore. Publication of the source code of the applications was not a requirement for both challenges. In addition, it is very likely that most applications would not have been able to make their source code available due to IP or funding restrictions.

3.3 Empirically-grounded Conceptual Architecture

Based on our empirical analysis, we now present a conceptual architecture for Linked Data applications. As defined by Soni et al. [497], a *conceptual architecture* describes a system in terms of its major design elements and the relationships among them, using design elements and relationships specific to the domain. It enables the discussion of the common aspects of implementations from a particular domain, and can be used as a high-level architectural guideline when implementing a single application instance for that domain.

Our conceptual architecture describes the high level components most often used among the surveyed applications to implement functionality that substantially differentiates applications using RDF or Linked Data from database-driven applications. For each component we describe the most common implementation strategies that are suited for specific application goals and deployment scenarios.

We first explain the choice of architectural style for our conceptual architecture, and we describe our criteria for decomposing the surveyed applications into components. Then we provide a detailed description of the components. This is followed by a discussion of the main implementation challenges that have emerged from the survey data. In addition, we suggest future approaches for applying the conceptual architecture.

3.3.1 Architectural Style of the Conceptual Architecture

Fielding [196] defines an architectural style as a coordinated set of architectural constraints that restricts the roles and features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.

Our empirical survey of RDF-based applications showed that there is a

TABLE 3.4: Results of the architectural analysis by year and component

year	number of apps	graph access layer	RDF store	graph-based nav. interface	data homog. service	graph query language service	structured data authoring interface	data discovery service
2003	10	100%	80%	90%	90%	80%	20%	50%
2004	16	100%	94%	100%	50%	88%	38%	25%
2005	6	100%	100%	100%	83%	83%	33%	33%
2006	19	100%	95%	89%	63%	68%	37%	16%
2007	24	100%	92%	96%	88%	88%	33%	54%
2008	23	100%	87%	83%	70%	78%	26%	30%
2009	26	100%	77%	88%	80%	65%	19%	15%
total	124	100%	88%	91%	74%	77%	29%	30%

range of architectural styles among the surveyed applications, which is dependent on the constraints of the respective software architects. For example, as existing Web service infrastructure can be reused as part of an application, many applications chose a *service-oriented architecture*. Applications that are focused on reusing existing databases and middleware infrastructure preferred a *layered style*. The *client-server architecture* was used when decentralized deployment across organizations or centralized storage were important. Applications that prioritized robustness, decentralization, or independent evolution favored a *peer-to-peer architecture*. Finally, the architecture of all other applications usually was defined by the reuse of existing tools and libraries as components in a *component-based architecture*.

As the component-based architectural style is the lowest common denominator of the surveyed applications, we chose the component-based architectural style for our conceptual architecture. It allows us to express the most common high-level functionality that is required to implement Semantic Web technologies as separate components.

3.3.2 Decomposing the Surveyed Applications into Components

In order to arrive at our component-based conceptual architecture, we started with the most commonly found functionality among the surveyed applications: a component that handles RDF, an integration component, and a user interface. With these components as a starting point, we examined whether the data from the architectural analysis suggested we split them into further components. Table 3.4 shows the results of this architectural analysis. The full data of the analysis is available online[4].

Every surveyed application (100%) makes use of RDF data. In addition, a large majority (88%), but not all surveyed applications implement persistent

[4] `http://preview.tinyurl.com/component-survey-results-csv`

storage for the RDF data. In practice many triple stores and RDF libraries provide both functionality, but there are enough cases where this functionality is de-coupled, e.g. if the application has no local data storage, or only uses SPARQL to access remote data. This requires splitting of the RDF-handling component into two components—first, a *graph access layer*, which provides an abstraction for the implementation, number, and distribution of data sources of an application; second, an *RDF store* for persistent storage of graph-based RDF data.

A query service which implements SPARQL, or other graph-based query languages, in addition to searching on unstructured data, is implemented by 77% of surveyed applications. Such high coverage suggested the need for a *graph query language service*. It is separate from the graph access layer, which provides native API access to RDF graphs.

A component for integrating data is implemented by 74% of the applications. It provides a homogeneous perspective on the external data sources provided by the graph access layer. Usually external data first needs to be discovered and aggregated before it can be integrated - an integration service would offer this functionality. However only 30% of the surveyed applications required this functionality. For this reason, we split the integration functionality into a *data homogenization service* and a *data discovery service*.

Most (91%) applications have a user interface. All user interfaces from the surveyed applications allow the user to navigate the graph-based data provided by the application. Only 29% provide the means for authoring new data. Thus the user interface functions were split between two components: the *graph-based navigation interface* and the *structured data authoring interface*.

3.3.3 Description of Components

The resulting seven components describe the largest common high-level functionality which is shared between the surveyed applications in order to implement the Linked Data principles and Semantic Web technologies. For each component we give a name, a description of the role of the component, and a list of the most common implementation strategies for each component. In addition, we list related chapters in this book for each component, as well as other related literature.

Figure 3.1 shows the component diagram for the example application in Section 3.5 using the components and connectors of our conceptual architecture, which are summarized in Table 3.5.

Graph Access Layer

Also known as data adapter or data access provider. This component provides the interface needed by the application logic to access local or remote data sources, with the distinction based on physical, administrative, or organizational remoteness. In addition this component provides a translation

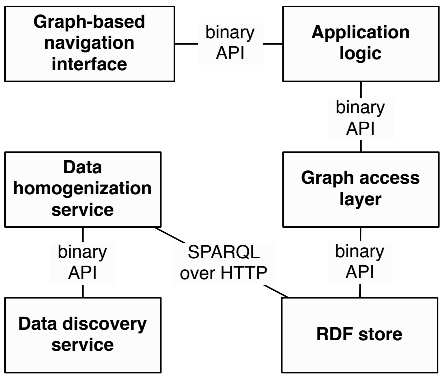

FIGURE 3.1: Component diagram for the example application in Section 3.5 using the components and connectors of our conceptual architecture

TABLE 3.5: Summary of components and connectors

Components	**Connectors**
Graph access layer	HTTP
RDF store	HTTPS
Data homogenization service	SPARQL over HTTP
Data discovery service	SQL connection
Graph query language service	binary API
Graph-based navigation interface	
Structured data authoring interface	

or mapping from the native data model of the programming language to the graph-based data model of RDF. All (100%) of the applications have a graph access layer.

This component is separate and distinct from the RDF store and graph query language service components, as it provides an abstraction layer on top of other RDF stores, query interfaces, and legacy document storage or relational databases.

Libraries for accessing RDF stores locally or remotely via SPARQL are available for all major programming and scripting languages. Oren et al. [422] describes the ActiveRDF approach for mapping between object oriented programming languages and the RDF data model. Another more recent approach to map between object oriented programming languages and the RDF data model is the Object triple mapping (OTM) described by Quasthoff and Meinel [446]. In Chapter 6, Calbimonte and Corcho discuss mapping the information generated from e.g. mobile devices and sensors to RDF, and querying the streams of subsequently generated Linked Data.

Implementation strategies: Accessing local data is implemented via programmatic access through RDF libraries by at least 50% of the applications; 47% use a query language for accessing local or remote data sources. Most of these applications use the SPARQL standard. Decentralised sources are used by 85%; 91% use data from multiple owners; 73% can import external data and 69% can export their data or provide a SPARQL end-point to make their data reusable; 65% of applications support data-updating during run-time.

RDF Store

Also known as triple store, persistent storage, or persistence layer. This component provides persistent storage for RDF and other graph-based data. It is accessed via the graph access layer; 88% of applications have such functionality.

The current state-of-the-art approaches for RDF storage, indexing, and query processing is described in Chapter 5, and Chapter 7 describes how to store RDF in the Amazon Web Services (AWS) cloud.

Bizer and Schultz [101] provide an overview of the features and the performance of RDF stores as part of the Berlin SPARQL Benchmark results. Notable examples of RDF stores include OpenLink Virtuoso, which is described in Chapter 9, the Bigdata RDF store, which is described in Chapter 8, as well as Jena [568] and the Jena TDB[5] backend.

Implementation strategies: Possible supported standards include, but are not limited to, data representation languages (XML, RDF), meta-modelling languages (OWL, RDFS) and query languages (SQL, SPARQL). RDF is explicitly mentioned by 96% of applications; OWL is supported by 43%; RDF Schema is supported by 20%. Inferencing or reasoning on the stored data is explicitly mentioned by 58% of the applications. Another implementation strategy is to use a relational database to store RDF data, which is described in more detail in Chapter 4.

Data Homogenization Service

Also known as integration service, aggregation service, mediation service, or extraction layer. This component provides a means for addressing the structural, syntactic, or semantic heterogeneity of data resulting from data access to multiple data sources with different formats, schemas, or structure. The service goal is to produce a homogeneous view on all data for the application. The data homogenization service often needs to implement domain or application specific logic. 74% of the applications implement this component.

[177] provides a general overview of semantic integration, and [152] provides an overview of using Semantic Web technologies for integration.

Chapter 2 goes into details of the current state-of-the-art in aligning ontologies for integrating Linked Data. In addition, Chapter 16 describes incre-

[5]`https://jena.apache.org/documentation/tdb/`

mental reasoning on RDF streams, which can be used for integrating Linked Data from different sources.

Implementation strategies: Integration of heterogeneous data is supported by 87% of the applications; 91% support data integration from sources with different ownership. Integration of data from distributed, decentralized data sources is supported by 85%. These three properties are orthogonal, as it would be possible, for example, to support just SIOC data, which is not heterogeneous, but which may be aggregated from personal websites, so that the data sources are distributed and under different ownership.

The four major styles of implementing the data homogenization service are: Automatic integration (11%), which is performed using heuristics or other techniques to avoid human interaction; semi-automatic integration (59%), which requires human inspection of the source data, after which rules, scripts, and other means are manually created to integrate the data automatically in the future; manual integration (10%) in which the data is completely edited by a human; finally, 20% do not need any data integration because they operate on homogeneous sources.

Data Discovery Service

Also known as crawler, harvester, scutter, or spider. This component implements automatic discovery and retrieval of external data. This is required where data should be found and accessed in a domain specific way before it can be integrated. 30% of applications implement a data discovery service.

Chapter 10 describes how to discover and access Linked Data from the Web as a distributed database. Harth et al. [258] introduce different research issues related to data crawling. Kafer et al. [313] provide an empirical survey of the change frequency of Linked Data sources. Chapter 15 describes how to represent and store provenance information for Linked Data. Finally, Chapter 17 describes the LInked Data Services (LIDS) approach which enables (semi-)automatic discovery and integration for distributed Linked Data services.

Implementation strategies: The component should support different discovery and access mechanisms, like HTTP, HTTPS, RSS. Natural language processing or expression matching to parse search results or other web pages can be employed. The service can run once if data is assumed to be static or continuously if the application supports updates to its data (65%).

Graph Query Language Service

Also known as query engine or query interface. This component provides the ability to perform graph-based queries on the data, in addition to search on unstructured data. Interfaces for humans, machine agents, or both can be provided. 77% of applications provide a graph query language service.

Mangold [378] provides a survey and classification of general semantic search approaches. Arenas and Pérez [44] provide an introduction to SPARQL and how it applies to Linked Data. Chapter 5 discusses optimization of SPARQL query processing and provides a survey of join and path traversal processing in RDF databases. Chapter 14 provides an overview of the state-of-the-art of the concepts and implementation details for federated query processing on linked data. Chapter 12 introduces data summaries for Linked Data, which enable source selection and query optimization for federated SPARQL queries. Finally, Chapter 13 describes peer to peer based query processing over Linked Data as an alternative approach.

Implementation strategies: Besides search on features of the data structure or semantics, generic full text search can be provided. An interface for machine agents may be provided by a SPARQL, web service or REST endpoint. SPARQL is implemented by 41% of applications, while the preceding standards SeRQL and RDQL are explicitly mentioned by only 6%.

Graph-based Navigation Interface

Also known as portal interface or view. This component provides a human accessible interface for navigating the graph-based data of the application. It does not provide any capabilities for modifying or creating new data. 91% of applications have a graph-based navigation interface.

Dadzie and Rowe [162] provide an overview of approaches used for visualizing Linked Data. One of the most popular JavaScript libraries for displaying RDF and other structured data as faceted lists, maps, and timelines is Exhibit [300]. The NautiLOD language for facilitating semantic navigation on Linked Data is described in Chapter 11.

Implementation strategies: The navigation can be based on data or metadata, such as a dynamic menu or faceted navigation. The presentation may be in a generic format, e.g. in a table, or it may use a domain specific visualization, e.g. on a map. Most navigation interfaces allow the user to perform searches, Hildebrand et al. [283] provide an analysis of approaches for user interaction with semantic searching.

Structured Data Authoring Interface

This component allows the user to enter new data, edit existing data, and import or export data. The structured data authoring component depends on the navigation interface component, and enhances it with capabilities for modifying and writing data. Separation between the navigation interface and the authoring interface reflects the low number of applications (29%) implementing write access to data.

The Linked Data Platform (LDP) specification, which enables accessing, updating, creating, and deleting resources on servers that expose their resources as Linked Data, is described in Chapter 18.

The Semantic MediaWiki project [346] is an example where a user interface is provided for authoring data. Another example is the OntoWiki [278] project.

Implementation strategies: The annotation task can be supported by a dynamic interface based on schema, content, or structure of data. Direct editing of data using standards such as e.g. RDF, RDF Schema, OWL or XML can be supported. Input of weakly structured text using, for example, wiki formatting can be implemented. Suggestions for the user can be based on vocabulary or the structure of the data.

3.4 Implementation Challenges

When comparing the development of database-driven and RDF-based applications, different implementation challenges arise that are unique to Linked Data and Semantic Web technologies. Based on our empirical analysis we have identified the most visible four challenges: Integrating noisy data, mismatched data models between components, distribution of application logic across components, and missing guidelines and patterns.

Identifying these challenges has two benefits. It allows practitioners to better estimate the effort of using RDF or Linked Data in an application. In addition, it allows tool developers to anticipate and mitigate these challenges in future versions of new libraries and software frameworks.

3.4.1 Integrating Noisy and Heterogeneous Data

One objective of RDF is to facilitate data integration [280] and data aggregation [96]. However, the empirical data from our analysis shows that the integration of noisy and heterogeneous data contributes a major part of the functional requirements for utiliuing RDF or Linked Data.

Our analysis demonstrates the impact of the integration challenge on application development: The majority (74%) of analyzed applications implemented a data homogenization service. However some degree of manual intervention was necessary for 69% of applications. This means that prior to integration, data was either manually edited or data from different sources was manually inspected in order to create custom rules or code. Only 11% explicitly mention fully automatic integration using e.g. heuristics or natural language processing. 65% allowed updating of the data after the initial integration, and reasoning and inferencing was also used for 52% of integration services.

3.4.2 Mismatch of Data Models between Components

The surveyed applications frequently had to cope with a software engineering mismatch between the internal data models of components. Accessing

RDF data from a component requires mapping of an RDF graph to the data model used by the component [422].

Most of the analyzed applications (about 90%) were implemented using object-oriented languages, and many of the analyzed applications stored data in relational databases. This implies that these applications often had to handle the mismatch between three different data-models (graph-based, relational, and object-oriented). This can have several disadvantages, such as introducing a high overhead in communication between components and inconsistencies for round-trip conversion between data models.

3.4.3 Distribution of Application Logic across Components

For many of the components identified by our analysis, the application logic was not expressed as code but as part of queries, rules, and formal vocabularies. 52% of applications used inferencing and reasoning, which often encode some form of domain and application logic; the majority of applications explicitly mentioned using a formal vocabulary, and 41% make use of an RDF query language. This results in the application logic being distributed across the different components.

The distribution of application logic is a well known problem for database-driven Web applications [356]. Current web frameworks such as Ruby on Rails[6] or the Google Web Toolkit[7] allow the application developer to control the application interface and the persistent storage of data programmatically through Java or Ruby, without resorting to a scripting language for the interface and SQL for the data storage. Similar approaches for centralizing the application logic of Linked Data applications still have to be developed.

3.4.4 Missing Guidelines and Patterns

Most Semantic Web technologies are standardized without providing explicit guidelines for the implementation of the standards. This can lead to incompatible implementations especially when the interplay of multiple standards is involved. The establishment of a community consensus generally requires comparison and alignment of existing implementations. However, the waiting period until agreed guidelines and patterns for the standard have been published can have a negative impact on the adoption of a standard. To illustrate this implementation challenge, we discuss the most visible instances from our survey data.

Publishing and consuming data: All of the applications consume RDF data of some form; 73% allow access to or importing of user-provided external data, and 69% can export data or can be reused as a source for another application. However the analysis shows that there are several incompatible

[6]`http://rubyonrails.org/`
[7]`https://developers.google.com/web-toolkit/`

possible implementations for publishing and consuming of RDF data. Only after the publication of the Linked Data principles [83] in 2006 and the best practices for publishing Linked Data [98] in 2007, did an interoperable way for consuming and publishing of data emerge. This is also reflected by the increasing use of these guidelines by the analyzed applications from 2007 on.

Embedding RDF on web pages: While the majority of applications in our survey have web pages as user interfaces, RDF data was published or stored separately from human readable content. This resulted in out-of-sync data, incompatibilities, and difficulties for automatically discovering data. Our analysis shows that after finalizing the RDFa standard[8] in 2008, embedding of RDF on web pages strongly increased.

Writing data to a remote store: While SPARQL standardized remote querying of RDF stores, it did not include capabilities for updating data. Together with a lack of other standards for writing or updating data, this has resulted in a lot of applications (71%) in our survey which do not include a user interface for authoring data. The lack of capabilities of creating, editing, or deleting data is addressed by SPARQL Version 1.1[9] in 2012.

Restricting read and write access: One capability that is currently mostly missing from all Semantic Web standards is the management of permissions for read or write access to resources and RDF stores. Our empirical analysis shows, that if such capabilities are required for an application, then they will be implemented indirectly via other means. We hope that this will be addressed in the near future by the successful standardization of current efforts such as WebID [510], which provides a generic, decentralized mechanism for authentication through the use of Semantic Web technologies. At the time of writing, the W3C is working on standardizing write access for Linked Data via the Linked Data Platform (LDP) working group[10], which is described in Chapter 18 of this book.

3.5 Analysis of an Example Application

The challenges of implementing RDF-based applications become apparent even for small applications. Figure 3.2 shows the architecture of an application from the authors' previous work, the SIOC explorer [104]. It aggregates content from weblogs and forums exposing their posts and comments as RDF data using the SIOC vocabulary [110]. This allows following multiple blogs or forums from a single application. The *application logic* and most parts of the application are implemented using the Ruby scripting language and the

[8]`http://www.w3.org/TR/rdfa-primer/`
[9]`http://www.w3.org/TR/sparql11-query/`
[10]`http://www.w3.org/2012/ldp/`

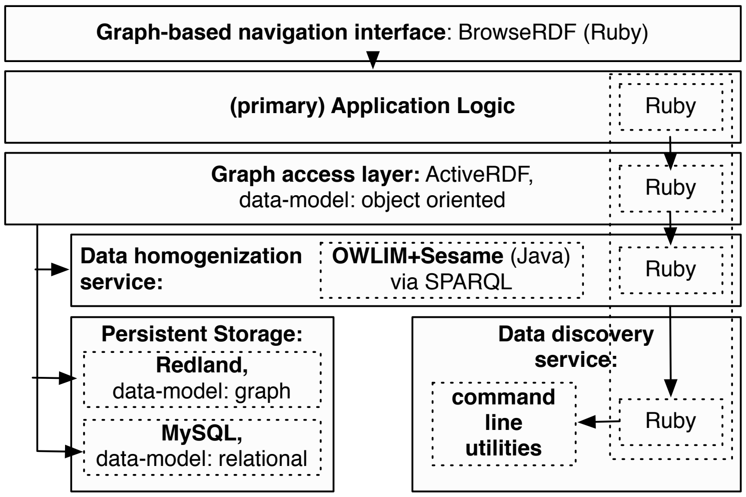
Graph-based navigation interface: BrowseRDF (Ruby)
(primary) Application Logic
Ruby
Graph access layer: ActiveRDF, data-model: object oriented
Ruby
Data homogenization service:
OWLIM+Sesame (Java) via SPARQL
Ruby
Persistent Storage:
Redland, data-model: graph
MySQL, data-model: relational
Data discovery service:
command line utilities
Ruby

FIGURE 3.2: Result of the architectural and functional analysis of an example application, the SIOC explorer

Ruby on Rails web application framework. The *graph-based navigation interface* allows faceted browsing of the SIOC data and is implemented through the BrowseRDF Ruby component [421]. The *graph access layer* is provided by ActiveRDF [422], which is an object-oriented Ruby API for accessing RDF data. The *data homogenization service* is implemented in two steps: (1) generic object consolidation is provided by the OWLIM extension for Sesame[11], written in Java, and accessed via SPARQL, and (2) domain specific integration of SIOC data is implemented as Ruby code. The Redland library[12] is used as an RDF store. Other application data (non-RDF data) is persistent to a MySQL relational database. A *data discovery service* is implemented through several Unix command line utilities which are controlled by Ruby. The SIOC explorer does not implement a graph query language service search service or a structured data authoring interface.

All four identified implementation challenges affect the SIOC explorer: (1) Even though all data sources use RDF and the SIOC vocabulary, the *data is noisy* enough to require two steps of data homogenization. The OWLIM extension of Sesame provides generic object consolidation, and integration specific to SIOC data is implemented as Ruby code. (2) The *components are mismatched*, regarding both the data models (object oriented, relational, and graph based) and the programming language APIs (Ruby and Java). This requires mapping RDF to Ruby objects (ActiveRDF) and mapping relational data to Ruby objects (ActiveRecord). Sesame has no Ruby API, so SPARQL

[11]`http://ontotext.com/owlim/`
[12]`http://librdf.org/`

is used to access Sesame, resulting in slow performance for large numbers of concurrent read operations. (3) *Unclear standards* and best practices affect the crawler implementation, as different SIOC exporters require different methods to discover and aggregate the SIOC data, as RDFa and GRDDL were not in wide use when the SIOC explorer was developed in 2007. (4) The *application logic is distributed* across the primary application logic component, the data interface, the rules of the integration service, and the code which controls the crawler.

3.6 Future Approaches for Simplifying Linked Data Application Development

We propose software engineering and design approaches to facilitate the implementation of Linked Data applications in the future, and which mitigate the identified implementation challenges. These approaches are: (1) guidelines, best practices, and design patterns; (2) software libraries; and (3) software factories.

All of these approaches can be used to provide ready-made solutions and lower the entry barriers for software that is consuming from or publishing to the Web of Data. In addition, the reuse of such existing solutions can enable greater interoperability between Linked Data applications, thus leading to a more coherent Web of Linked Data.

3.6.1 More Guidelines, Best Practices, and Design Patterns

The Linked Data community has already developed some guidelines and collections of best practices for implementing Semantic Web technologies. These include best practices for publishing of Linked Data [275] and the naming of resources [465], and the Linked Data principles themselves.

In the future this can be complemented by implementation-oriented guidelines such as *design patterns*. A design pattern is a description of a solution for a reoccurring implementation problem [208]. A first collection of implementation-oriented design patterns is provided by the Linked Data patterns collection[13] , while [404] provides guidelines for requirements engineering and design of Linked Data applications.

3.6.2 More Software Libraries

In order to go beyond libraries for accessing and persisting of RDF data, more software libraries in the future will directly provide reusable implemen-

[13]http://patterns.dataincubator.org/book/

tations of guidelines and patterns. Several such libraries are currently being developed such as ActiveRDF, which maps the native data model of a programming language to RDF [422], and the SIOC PHP API[14] for implementing access to SIOC data. Best practices for converting relational databases to RDF are implemented in the D2R server[15], while any23[16] implements best practices for conversion of many structured data formats to RDF. The Silk framework[17] can be used for interlinking data sets. Additional projects for encoding best practices can be found on the LOD community wiki[18].

3.6.3 Software Factories

Software factories provide the means to create complete and fully functioning applications by building on patterns, best practices, and libraries [239]. They allow the assembly of complete applications from existing components and libraries that implement community best practices and patterns. Customization for a domain or a specific application is possible through predefined extension points and hooks.

While the majority of analyzed applications uses RDF libraries, some components (e.g. navigation interface, homogenization, and data discovery services) are usually custom made for each application. A software factory for Linked Data applications could provide a pre-made solution for each of the components from our conceptual architecture. Then the application developer could add the application and domain-specific logic and customize each of the components. This would allow rapid assembly of Linked Data applications. Similar benefits are already provided by modern Web application frameworks such as Ruby on Rails, PHPCake, and Django. [421] presented a first prototype of such a software factory for Linked Data applications.

3.7 Related Work

We now discuss related work in the areas of empirical surveys about the adoption of Semantic Web technologies and architectures for Semantic Web applications.

[14]http://sioc-project.org/phpapi
[15]http://d2rq.org/d2r-server/
[16]http://any23.org/
[17]http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/
[18]http://preview.tinyurl.com/LOD-wiki

3.7.1 Empirical Surveys

Cardoso [121] presents the results of a survey of 627 Semantic Web researchers and practitioners carried out in January 2007. The questions from the survey covered the categories of demographics, tools, languages, and ontologies. The goal of the survey was to characterize the uptake of Semantic Web technologies and the types of use-cases for which they were being deployed. Concrete applications were not taken into account. As the survey was carried out over a two months period of time, it does not allow conclusions about long term trends before or after the survey was taken.

Another similar survey of 257 participants (161 researchers and 96 industry-oriented participants) was published online[19] in 2009. The data for this survey was taken from a 3 month period. This survey had the goal of measuring the general awareness of Semantic Web technologies and social software in academia and enterprise. As such, the questions did not go into any implementation specific details.

Cunha [161] performed a survey of 35 applications from the "Semantic Web challenges" in 2003, 2004, and 2005. The goal of the survey was to cluster the applications based on their functionality and their architecture. The result is a list of 25 application categories with the number of applications per category for each year. This survey covers only 3 years, and it is not concerned with the adoption of Semantic Web technologies and capabilities, whereas our empirical survey provides empirical data on the uptake of e.g. data integration or the Linked Data principles.

3.7.2 Architectures for Semantic Web Applications

García-Castro et. al [209] propose a component-based framework for developing Semantic Web applications. The framework consists of 32 components aligned on 7 dimensions, and is evaluated in 8 use-cases. While existing software tools and libraries that implement these components are identified, the identification of the components is not based on an empirical grounding.

Tran et. al [530] propose a layered architecture for ontology-based applications that is structured in a presentation layer, logic layer, and data layer. It is based on best practices for service-oriented architecture and on the authors' model of life-cycle management of ontologies, and evaluated in a use-case. However there is no empirical grounding for the architecture.

A similar approach is taken by Mika and Akkermans [390] who propose a layered architecture for ontology-based applications, with layers for the application, the middle-ware, and the ontology. The architecture is based on a requirements analysis of KM applications.

Cunha [160] builds on his earlier survey in [161] and presents a UML architecture based on the 35 analyzed applications. The goal of the architecture

[19] `http://preview.tinyurl.com/semweb-company-austria-survey`

is to provide the foundation for a potential software framework, but no evaluation or implementation of the architecture is provided.

3.8 Conclusions

In this chapter we focused on the software architecture and engineering issues involved in designing and deploying Linked Data applications. We argue that software engineering can unlock the full potential of the emerging Web of Linked Data through more guidelines, best practices, and patterns, and through software libraries and software factories that implement those guidelines and patterns.

Greater attention needs to be given to standardizing the methods and components required to deploy Linked Data in order to prevent an adoption bottleneck. Modern Web application frameworks such as Ruby on Rails for Ruby, PHPCake for PHP, and Django for Python provide standard components and architectural design patterns for rolling out database-driven Web applications. The equivalent frameworks for Linked Data applications are still in their infancy.

As a guide to developers of Linked Data applications, we present an architectural analysis of 124 applications. From this we put forward a conceptual architecture for Linked Data applications, which is intended as a template of the typical high level components required for consuming, processing, and publishing Linked Data. Prospective developers can use it as a guideline when designing an application for the Web of Linked Data. For experienced practitioners it provides a common terminology for decomposing and analyzing the architecture of a Linked Data application.

ACKNOWLEDGEMENTS: *The work presented in this paper has been funded by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2). We also thank everybody from the Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, for their feedback and support, especially Laura Dragan and Maciej Dabrowski.*