Lessons and requirements learned from a decade of deployed Semantic Web applications[⋆]

Benjamin Heitmann, Richard Cyganiak, Conor Hayes, and Stefan Decker

firstname.lastname@deri.org
Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland

Explanation of interest in the workshop The CfP of the Linked Enterprise Data Patterns workshop, calls for submissions to air requirements, present patterns and to identify gaps in the current Linked Data standards and guidelines. Our contribution is based on an empirical survey of 124 Semantic Web applications of the last decade. This allows us to describe the current state of the art for developing and deploying applications on the Web of Data. We use this to present a component-based, conceptual architecture which provides an architectural pattern for applications using Linked Data. We also discuss important implementation challenges which where found as part of the empirical survey. These challenges identify gaps and new requirements in current Linked Data guidelines and standards. Finally, we suggest future approaches to facilitate the standardisation of components and the development of software engineering tools to increase uptake of the Web of Data. We suggest that our contributions can add an important empirical grounding for the discussion of requirements and gaps at the workshop.

1 Introduction

To date, investigating Semantic Web technologies and the Web of Data in terms of their cost of implementation complexity has not been given the same priority as the research on potential benefits.

In this article we use empirical software engineering as a research toolkit to summarise the lessons and requirements which can be learned from the first decade of deployed Semantic Web applications. We first discuss the results of an *empirical survey* of Semantic Web applications over most of the past decade (in section 2). This allows us to describe the current state of the art for developing and deploying Semantic Web applications. It also allows us to determine how far the adoption of signature research topics of the Semantic Web, 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 Semantic Web applications (in section 3). Our conceptual architecture describes the high level components which are most often observed among the participants of the survey. These components implement the functionality that differentiates applications using RDF from database-driven applications. 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.

Based on the empirical survey and the conceptual architecture, we discuss the main *implementation challenges* facing developers using Semantic technologies (in section 4). In addition we suggest *future approaches* in using the conceptual architecture to facilitate the standardisation of components and the development of software engineering tools to increase the uptake of the Web of Data (section 5).

The full article contains the following additional content: all original tables and figures; a short discussion of background terms; full descriptions for all seven components of the conceptual architecture; a discussion of threats to the validity of our empirical survey; as well as a list of related work.

2 Empirical survey

The evolution of Semantic Web technologies is characterised by the constant introduction of new ideas, standards and technologies, and thus appears to be in a permanent state of flux. However, nearly a decade has passed since Tim Berners-Lee et al. published their comprehensive vision for a Semantic Web [2] in 2001. This allows us to look back on the empirical evidence left behind by the Semantic Web applications that have been developed and deployed during this time.

Sjoberg et. al [10] 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 Semantic Web applications. Without empirical evidence it is difficult to evaluate the adoption rate of Semantic Web technologies. To provide such insights, we collect evidence from a representative sample of the population of developers deploying Semantic Web technologies. In particular we perform an empirical survey of over a hundred Semantic Web applications from two demonstration challenges that received much attention from the Semantic Web research community.

The results of our survey enable us to determine the state of adoption of key Semantic Web research goals such as data reuse, data integration and reasoning.

[⋆] This paper is an abbreviated version of Heitmann et al., "An empirically grounded conceptual architecture for applications on the Web of Data", IEEE Transations on Systems, Man and Cybernetics – Part C: Applications and Reviews, 2011 (DOI: 10.1109/TSMCC.2011.2145370). It is funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2).

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] [7] in the years 2003 to 2009 with 101 applications, which is organised 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 organised as part of the European Semantic Web Conference. 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.

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 which 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 decentralised 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: Firstly, 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 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.

2.2 Findings

The results of our empirical survey show that the adoption of the capabilities that characterise Semantic Web technologies has steadily increased over the course of the demonstration challenges.

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 data became available as the Web of Data expanded.

The simplification of data integration is claimed as one of the central benefits of implementing Semantic Web technologies. The results of the survey 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 which 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 data sources from the Web of Data that do not need to be integrated.

Support for decentralised sources, data from multiple owners, and sources with heterogeneous formats is supported each year by a large majority of the 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 then half of the contest entries in 2009, after being formulated in 2006 by Tim Berners-Lee [1]. Interestingly the increasing usage of the Linking Open Data (LOD) 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 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 summarise: Support for features which differentiate RDF-based from database-driven applications is now widespread amongst 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

[1] `http://challenge.semanticweb.org/`
[2] `http://www.semanticscripting.org`
[3] `http://the-blank.net/semwebappsurvey/`

reached more then half of the contest entries in 2009. And while the choice of implemented standards has crystallised 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 Conceptual architecture

Based on our empirical analysis, we now present a conceptual architecture for Semantic Web applications. As defined by Soni et al. [11], 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 RDF-supported applications from database-driven applications. It can be seen as evidence of a convergence to a common set of components for implementing Semantic Web applications.

3.1 Decomposing the surveyed applications into components

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.

In order to arrive at our component-based conceptual architecture, we started with the most commonly found functionality amongst 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. 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 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.

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

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 homogenisation 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*.

The full paper contains the full descriptions for all components, including common implementation strategies, references and examples.

4 Implementation challenges

When comparing the development of database-driven and RDF-based applications, different implementation challenges arise that are unique to Semantic Web technologies. Based on our empirical analysis we have identified the most visible four challenges. Identifying these challenges has two benefits. It allows practitioners to better estimate the effort of implementing Semantic Web technologies. In addition, it allows tool developers to anticipate and mitigate these challenges in future versions of new libraries and software frameworks.

Integrating noisy and heterogeneous data One objective of RDF is to facilitate data integration [6] and data aggregation [3]. 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 utilising Semantic Web technologies and data.

Our analysis demonstrates the impact of the integration challenge on application development: The majority (74%) of analysed applications implemented as a data homogenisation 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 the different sources was 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.

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.

Most of the analysed applications (about 90%) were implemented using object-oriented languages, and many of the analysed 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.

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 [8]. Current web frameworks such as Ruby on Rails or the Google Web Toolkit 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 centralising the application logic of Semantic Web applications still have to be developed.

Missing guidelines and patterns Most Semantic Web technologies are standardised 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 explain 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 possible implementations for publishing and consuming of RDF data. Only after the publication of the Linked Data principles [1] in 2006 and the best practices for publishing Linked Data [4] 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 analysed 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 finalising the RDFa standard in 2008, embedding of RDF on web pages strongly increased.

Writing data to a remote store: While SPARQL standardised 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%) which do not include a user interface for authoring data. This is being addressed in the next versions of SPARQL which will include data editing and updating capabilities.

Restricting read and write access: One capability that is currently mostly missing from all Semantic Web standards is the authorisation 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. This will be addressed by future standards: FOAF+SSL [12] provides a generic, decentralised mechanism for authentication through the use of Semantic Web technologies. This is complemented by RDF Push [13], which describes how to authenticate updates to RDF stores.

5 Approaches for simplifying Semantic Web application development

We propose software engineering and design approaches to facilitate the implementation of Semantic Web technologies in the future, and 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 Semantic Web applications, thus leading to a more coherent Web of Data.

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 publishing of Linked Data [4] and the naming of resources [9], and the Linked Data principles themselves. In the future this can be complemented by implementation-oriented guidelines such as patterns. A design pattern is a description of a solution for a reoccurring implementation problem. A first collection of implementation-oriented patterns is provided by the Linked Data patterns collection[5].

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 implementations of guidelines and patterns. Best practices for converting relational databases to RDF are implemented in the D2R server, while any23 implements best practices for conversion of many structured data formats to RDF. Various guidelines for accessing RDF are implemented by the Semantic Web Client Library. Additional projects for encoding best practices can be found on the LOD community wiki[6].

Software factories Software factories provide the means to create complete and fully functioning applications by building on patterns, best practices and libraries [5]. They allow the assembly of complete applications from existing components and libraries that implement community best practices and patterns. Customisation for a domain or a specific application is possible through pre-defined extension points and hooks. While the majority of analysed applications uses RDF libraries, some components (e.g. navigation interface, homogenisation and data discovery services) are usually custom made for each application.

A software factory for Semantic Web 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 customise each of the components. This would allow rapid assembly of Semantic Web applications. Similar benefits are already provided by modern Web application frameworks such as Ruby on Rails, PHPCake and Django.

6 Conclusion

In this article we have summarised the lessons and requirements which can be learned from the first decade of deployed Semantic Web applications. All our findings are based on empirical data using empirical software engineering as a research methodology.

The results of our *empirical survey* show that the adoption of the capabilities that characterise Semantic Web technologies and which differentiate RDF-based from database-driven applications is now widespread amongst the development communities of the challenges. There is significant adoption of signature research topics of the Semantic Web, such as data reuse, data integration and reasoning.

Our *conceptual architecture* describes the high level components which are most often observed among the participants of the survey: graph access layer, RDF store, data homogenisation service, graph query language service, graph-based navigation interface, and the structured data authoring interface. These components implement the functionality that differentiates applications using RDF from database-driven applications. 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.

Based on the empirical survey and the conceptual architecture, we have discussed the main *implementation challenges* facing developers using Semantic technologies: Integrating noisy data, mismatched data models between components, distribution of application logic across components and missing guidelines and patterns.

In addition we suggest *future approaches* to support the growth of the emerging Web of Data through software engineering. This includes fostering community consensus and documenting more best practices, guidelines and design patterns. The development of more software libraries will provide reusable implementations of best practices and patterns, and software factories can facilitate the standardisation of components for applications on the Web of Data.

[5] `http://patterns.dataincubator.org`

[6] `http://preview.tinyurl.com/LOD-wiki`

References

1. T. Berners-Lee. Linked Data - Design Issues. http://www.w3.org/DesignIssues/LinkedData.html, 2006.
2. T. Berners-Lee, J. A. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
3. C. Bizer. The Emerging Web of Linked Data. *IEEE Intelligent Systems*, pages 87–92, 2009.
4. C. Bizer, R. Cyganiak, and T. Heath. How to Publish Linked Data on the Web. Technical report, FU Berlin, 2007.
5. J. Greenfield and K. Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *Conference on Object-oriented programming, systems, languages, and applications*, 2003.
6. J. Hendler. Web 3.0 Emerging. *IEEE Computer*, 42(1):111–113, 2009.
7. M. Klein and U. Visser. Guest Editors' Introduction: Semantic Web Challenge 2003. *IEEE Intelligent Systems*, pages 31–33, 2004.
8. A. Leff and J. Rayfield. Web-application development using the Model/View/Controller design pattern. *International Enterprise Distributed Object Computing Conference*, pages 118–127, 2001.
9. L. Sauermann, R. Cyganiak, and M. Volkel. Cool URIs for the Semantic Web. W3C note, W3C, 2008.
10. D. I. K. Sjoberg, T. Dyba, and M. Jorgensen. The Future of Empirical Methods in Software Engineering Research. *Future of Software Engineering*, pages 358–378, 2007.
11. D. Soni, R. L. Nord, and C. Hofmeister. Software architecture in industrial applications. *International Conference on Software Engineering*, 1995.
12. H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF+SSL: RESTful Authentication for the Social Web. In *Workshop on Trust and Privacy on the Social and Semantic Web*, 2009.
13. O.-E. Ureche, A. Iqbal, R. Cyganiak, and M. Hausenblas. Accessing Site-Specific APIs Through Write-Wrappers From The Web of Data. In *Workshop on Semantics for the Rest of Us*, 2009.