Implementing Semantic Web applications: reference architecture and challenges

Benjamin Heitmann, Sheila Kinsella, Conor Hayes, and Stefan Decker

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

Abstract. To date, Semantic Web research has tended to focus on data modelling challenges, at the expense of software architecture and engineering issues. Our empirical analysis shows that implementing Semantic Web technologies creates challenges which can affect the whole application. Standard solutions and best practices for Semantic Web technologies are just emerging. The lack of these has been an obstacle for implementing and deploying applications which exploit Semantic Web technologies for real world use cases.

In this paper we conduct an empirical survey of Semantic Web applications. We use this empirical data to propose a reference architecture for Semantic Web applications, and to identify the four main challenges for implementing the most common functionality related to Semantic Web technologies from a software engineering perspective: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. We describe two orthogonal approaches for mitigating these challenges: (a) simplifying the application architecture by delegating generic functionality to external service providers, and (b) assembling and customising of components provided by software frameworks for rapid development of complete applications.

1 Introduction

Semantic Web technologies simplify knowledge-intensive applications, by enabling a Web of interoperable and machine-readable data [1] based on formal and explicit descriptions of the structure and semantics of the data [2].

Existing research on deploying Semantic Web technologies has tended to focus on data modelling, and software architecture and engineering issues have been comparatively neglected. Benefits such as simplification of information retrieval [3], information extraction [4] and data integration [5] have been well researched.

However, in order to encourage wide scale adoption of Semantic Web technologies, the whole life cycle of Semantic Web data needs to be assessed in terms of efforts and pay back of the application development. According to [6] this life cycle includes: the initial phase of *ontology development*, followed by *planning* how to use the data, *creation* of new data or *refining* of existing data, then persistent *archiving* of data, and finally *publication* and external *access* of data. Creation,

refining, archiving and publication may all be performed at runtime by the application, and as such involve aspects of software engineering and software architecture in addition to data modelling aspects.

While the challenges of ontology development have been analysed based on empirical data of ontology development projects [7], to our best knowledge no empirical analysis of the challenges involved in the implementation of creating, refining, archiving and publication of data based on Semantic Web technologies exists.

We have performed an empirical survey of 98 Semantic Web applications (Section 2), which allows us to identify the most common shared components and the challenges in implementing these components. Together, these components constitute a reference architecture for Semantic Web applications. The survey shows that implementing Semantic Web technologies creates challenges which can affect the whole application. Standard solutions and best practices for Semantic Web technologies are just emerging. The lack of these has been an obstacle for implementing and deploying applications which exploit Semantic Web technologies for real world use cases.

Based on the survey, we identify the four main challenges (Section 3) for implementing Semantic Web applications: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. Identifying these challenges allows better assessment of the costs associated with adopting Semantic Web technologies within enterprises, and forms the basis for designing better software frameworks and software architecture for exploiting the emerging Web of Data.

Towards this goal, we present two approaches for mitigating the identified challenges (Section 4) from a software engineering perspective. The first approach proposes an architectural solution by delegating generic components to external service providers thus simplifying the application. An orthogonal approach is to provide better software engineering support with components provided by software frameworks for rapid assembling and customising of complete applications. Finally, we list related research (Section 5) and discuss future research (Section 6).

The main contributions of this paper are (1) an empirical analysis of the state of the art regarding the implementation of the most common components of Semantic Web applications, (2) a reference architecture for Semantic Web applications based on the empirical analysis, (3) identifying the main challenges in implementing these components which are introduced by Semantic Web technologies, and (4) two approaches to mitigate these challenges from a software engineering perspective.

2 Empirical analysis of Semantic Web applications

As our goal is to identify the main challenges introduced by implementing Semantic Web technologies, we have performed an empirical analysis of the most common capabilities specific to Semantic Web applications. In Section 2.1, we provide a classification for Semantic Web applications in order to differentiate them from other applications on the World Wide Web. Section 2.2 outlines the methodology of the survey. The results of our survey follow in Section 2.3. First, we present a description of components which abstract the most common functionality related

to Semantic Web technologies. Secondly, we provide statistics about the variations amongst the implementations of the components.

2.1 Classifying Semantic Web applications and the Web of Data

The most basic requirement for a Semantic Web application is the use of RDF for the metadata used by the application. This can be derived from the fundamental role of RDF in the "layer cake" of Semantic Web standards [8]. Additionally a set of formal vocabularies should be used to capture the application domain, and `SPARQL` should be used as data query language, according to [9, definition 2.2]. All surveyed applications meet these requirements, except for applications using programmatic access to RDF data for efficiency reasons.

The Linked Data principles define how to publish RDF data, so that RDF data sets can be inter-linked [10] to form a Web of Data. The Linking Open Data community project(`http://linkeddata.org`) provides most of the currently available linked data.

2.2 Methodology of the survey

The survey of current Semantic Web applications has been performed in two parts, consisting of an architectural analysis and a questionnaire about the application functionality.

Architectural analysis The applications from two key demonstration challenges in the Semantic Web domain have been analysed to identify the most common functionality of Semantic Web applications: the "Semantic Web challenge"(`http://challenge.semanticweb.org/`), organised as part of the International Semantic Web Conference from 2003 to 2008, and the "Scripting for the Semantic Web challenge"(`http://www.semanticscripting.org`), organised as part of the European Semantic Web Conference from 2006 to 2008. Duplicate submissions have been eliminated, resulting in a total number of 98 surveyed applications.

The result of the architectural analysis is a list of components which provide an abstraction of the most common functionality which is required to implement Semantic Web standards. The components have been extracted from the architecture diagrams and the textual descriptions of the application architecture and implementation, depending on availability in the submitted paper. The components provide a common way to decompose the surveyed applications, so that components with similar functionality from different applications can be compared. This allows us to e.g. identify the need for data updating standards in section 3.3, as most applications have a user interface, but only a minority of applications allow creation of new data by the user.

Application functionality questionnaire Additionally a questionnaire was used to collect details about the implementation of the applications. The questionnaire contains 27 properties associated with 7 areas of functionality. The results from the questionnaire provide statistics about the range of variations in which the functionality of the common components has been implemented.

The questionnaire covers these areas of functionality: (1) implementation of Semantic Web standards, (2) support for data sources, (3) support for formal vocabularies that are heterogeneous and have diverse ownership, (4) implementation of data integration and alignment, (5) support for structured, semi-structured, unstructured or multimedia data, (6) support for authoring and editing of data, and (7) support for external data sources and the open-world assumption.

Only the applications from the "Semantic Web challenge" 2003 to 2006, and the "Scripting for the Semantic Web challenge" 2005 to 2007 were analysed with the questionnaire. The authors of the papers describing the applications where asked to validate and correct the details about their applications. Of the 50 applications analysed with the questionnaire, 74% validated their data.

2.3 Survey results

Taken together, the two parts of the survey can be combined to provide an overview of the state of the art in implementing the required functionality for Semantic Web technologies. The architectural analysis provides a list of the most common components, and the questionnaire provides statistical data about the different variations of implementing each component.

Table 1 shows the seven most common components, and lists the number of applications implementing a specific component by year.

year	number of applications	data interface	persistence storage	user interface	integration service	search service	authoring interface	crawler
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%
total	98	100%	91%	92%	72%	81%	32%	35%

Table 1. Percentage of surveyed applications implementing the 7 most common components, per year and in total

2.4 Reference architecture for Semantic Web applications

The surveyed applications share a significant amount of functionality regarding common capabilities of Semantic Web applications. We abstract from the differences between individual applications and distinguish seven main components, which together constitute a reference architecture for Semantic Web applications by describing high-level concepts and terminology, without fixing interfaces [11, page 242].

The (i) **data interface** provides an abstraction over remote and local data sources, the (ii) **persistent storage** stores data and run time state, and the (iii) **user interface** provides access for the user. (i) to (iii) have each been implemented by **more than 90%** of surveyed applications. The (iv) **integration service** provides a unified view on heterogeneous data, and the (v) **search service** allows searching in data. (iv) and (v) have each been implemented by **70% to 80%** of surveyed applications. The (vi) **crawler** discovers and retrieves remote data, and the (vii) **authoring interface** allows creating new data and editing existing data. (vi) and (vii) have each been implemented by **30% to 40%** of surveyed applications.

In the following we describe the functionality of each component in detail and provide statistical data for the range of variations amongst the implementations of the surveyed applications. The full results of the architectural analysis are available on-line(`http://semwebapp-components.dabbledb.com/`), as are the details of the questionnaire results(`http://www.activerdf.org/survey/`).

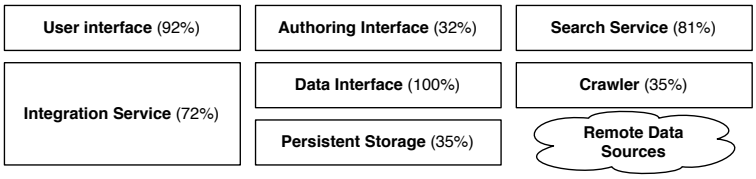

Fig. 1. The reference architecture for Semantic Web applications, with the percentage of surveyed applications implementing the component

Data Interface: *Also known as data adapter or data access provider.* Provides the **interface** needed by the application logic to **access local or remote data sources**, with the distinction based on either physical remoteness or administrative and organisational remoteness. Separation from the persistence layer is motivated by the function of the data interface as an **abstraction layer** regarding the implementation, number and distribution of persistence layers. 100% of the applications have a data interface.

Component variations: Accessing local data is implemented via programmatic access through RDF libraries by at least 50% of the applications. Only 24% use a query language for accessing local or remote data sources, but only half of these applications use the SPARQL standard. Multiple data sources with different ownership are used by 90% of applications, 70% support external data provided by the user and 60% can export their data or make it reusable as a source for other applications, by e.g. providing a SPARQL end-point. 76% of the applications support updating their data during application runtime.

Persistent Storage: *Also known as persistence layer or triple store.* Provides persistent **storage for data and run time state** of the application, it is accessed via the data interface. In practice many triple stores and RDF libraries provide both a data interface and persistent storage, but there are cases where the components are de-coupled, e.g. if the application has no local data storage, and only uses SPARQL to access remote data. 91% have a persistent storage.

Component variations: 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 86% of applications, OWL is supported by 48%, RDFS by 22%. Inferencing or reasoning on the stored data is explicitly mentioned by 58% of the applications. Storage of any combination of structured, semi-structured, unstructured data or (binary) files can be implemented, with different levels of features or optimisation for the different data types. 58% implement support for unstructured text and 48% support mixing of structured and unstructured data in some way.

User Interface: *Also known as portal interface or view.* Provides a **human accessible interface** for using the application and **viewing the data**. Does not provide any capabilities for modifying or creating new data. 92% have a user interface, as some applications do not provide a human usable interface.

Component variations: 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 visualisation, e.g. on a map (10%). 16% present images to the user and 6% explicitly mention support for audio content in the user interface. 28% support multiple languages in the user interface, thus catering to a multilingual audience.

Integration Service: *Also known as integration, aggregation, mediation, extraction layer or service.* Provides means for **addressing structural, syntactic or semantic heterogeneity** of data, caused by accessing data from multiple data sources using diverse kinds of format, schema or structure. The desired result is a **homogeneous view on all data** for the application. The integration service often needs to implement domain or application specific logic for the data integration. 72% of the applications have an integration service.

Component variations: Integration of heterogeneous data is supported by 90% of the applications, and 90% support data from sources with different ownership. Data from distributed data sources is supported by 72%. These three properties are orthogonal, as it would be e.g. possible to support just SIOC data [12] which is not heterogeneous, but which is aggregated from personal websites, so that the data sources are distributed and under different ownership.

Mapping or alignment between different schema may be automatic (12%), but most applications (80%) require some form of human intervention for the integration. Reasoning and inferencing can be used for the integration (58%). Integration may be performed once if data stays static, or continuously if new data gets added.

Search service: *Also known as query engine or query interface.* Provides the ability to **perform searches** on the data based on the content, structure or domain specific features of the data. **Interfaces for humans, machine agents or both** can be provided. 81% provide a search service.

Component variations: Besides search on features of the data structure or semantics, generic full text search (58%) or a search on unstructured and structured data at the same time (48%) can be provided. The interface for machine agents may be provided by e.g. a SPARQL, web service or REST endpoint.

Crawler: *Also known as harvester, scutter or spider.* Required if data needs to be found and accessed in a domain specific way before it can be integrated. Implements **automatic discovery and retrieval of data**. 35% implement a crawler. Some applications have an integration service, but do not need a crawler, e.g. because they only use local RDF data, but need to perform object consolidation [13].

Component variations: Support of 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 crawler can be active once if data is assumed to be static or continuous (76%) if new data needs to be discovered.

Authoring interface: Allows the user to **enter new data, edit existing data, and import or export data**. This component depends on the user interface component, and enhances it with capabilities for modifying and writing data. Separation between the user interface and the authoring interface reflects the low number of applications (32%) implementing write access to data.

Component variations: 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 e.g. wiki formatting can be implemented. Suggestions for the user can be based on vocabulary or the structure of the data.

3 The main challenges for implementing Semantic Web technologies

The list of common components and the data about the variations in implementing these components allow us to identify the main challenges for Semantic Web application development: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. In the following we detail these challenges and subsequently explain their impact on an example application.

3.1 Integrating noisy and heterogeneous data

An objective of RDF is to facilitate data integration [5] and aggregation [4]. However, even if all data sources were to use RDF as their data model, there would still exist potential integration issues due to different access mechanisms, noisy and erroneous data, and inconsistent usage of vocabularies and instance URIs between sources. Therefore, depending on how noisy and disconnected the data is, some amount of pre-processing may be required before using the data in an application.

Our survey shows that implementing integration of noisy and heterogeneous data can contribute the biggest part of the application functionality required for utilising Semantic Web technologies. The majority (72%) of surveyed applications implement an integration service. However manual intervention as part of the integration is necessary for 80% of applications. This means that prior to integration, either data is manually edited or data from the different sources is inspected in order to create custom rules or code. Only 20% explicitly mention fully automatic integration using e.g. heuristics or natural language processing. 76% allow updating of the data after the initial integration, and reasoning and inferencing is used for 58% of integration services.

Semantic Web data may be accessible by multiple different methods: large dumps to be downloaded, individual (possibly dynamically-generated) documents which need to be crawled, or via SPARQL endpoints to which queries must be issued. In order to ease the acquisition of published data, site suppliers can provide a semantic sitemap [14] on their website, so that crawling agents know where to find related RDF data. There are also a set of best practice guidelines [10] for publishing and interlinking pieces of data on the Semantic Web. The creation of ontologies is beyond the scope of this paper but has been discussed in previous literature [7].

Previous research performed using the Swoogle system [15] shows that there is a spectrum for RDF data, ranging from well structured data using stable and slowly changing vocabularies and ontologies to noisy and dynamic data with undefined terms and formal errors. The study tracked size changes in three versions of 183k RDF documents, and found changes had occurred in 60% of these documents. It also found that 2.2% of terms had no definitions and some had both class and property meta-usage. The Swoogle study also showed that the size distribution of Semantic Web documents is highly skewed, with many small documents and much fewer very large documents. Similarly, a study [16] using the WATSON infrastructure concludes that the Semantic Web is composed of many small, lightweight ontologies and fewer large, heavyweight ontologies. Assuming that within an ontology there is generally consistent use of vocabularies and instance URIs, the large number of smaller lightweight ontologies will present more problems for data integration.

The Semantic Web bug tracker(`http://bugs.semanticweb.org/`) is an initiative to improve the quality of the Web of data by tracking issues which could introduce inaccuracies in systems consuming the data. Previous work [17, 13] by the authors has also shown up various inconsistencies in RDF data on the Web. Some examples of frequent errors occurring in Semantic Web data observed from these sources are:

- **Use of non-standard terms:** There is frequent usage of classes and properties which are not defined in the official specifications. A study of ontology usage [17] shows over 1m definitions of instances of the class `foaf:chatEvent`, which does not exist in the official FOAF specifications(`http://xmlns.com/foaf/0.1/`).
- **Incorrect usage of vocabularies:**
  Publishers frequently use terms from vocabularies in ways which they were not intended and which may introduce unexpected results after reasoning. For example, dbpedia, which publishes structured data extracted from Wikipedia, uses the property `foaf:img` to link resources of all types to associated images. This is problematic because according to the FOAF specifications, the property `foaf:img` has a domain of `foaf:Person`. This means that confusingly, a reasoning system could infer that all Dbpedia resources with images are of type `foaf:Person`.
- **Multiple URIs for the same objects:** The ability to uniquely identify arbitrary resources via URIs is an important factor in data integration. However there is little agreement between sources on which URIs to use for a particular resources. This is a problem as it may result in potentially useful information about a resource being missed. Reasoning on inverse functional properties (IFPs) can alleviate this to some extent. An evaluation of an object consolidation algorithm [13] showed that 2.4 million instances could be consolidated into 400k instances. However noise in IFP statements can cause even more problems. [13] notes that in filling out online profiles, users who do not wish to reveal their instant messaging usernames will fill in an alternative value such as "none". As a result, 85k of the users supplying these non-unique usernames were incorrectly merged.

The majority of applications rely on data integration, but in order to implement it, expensive human intervention is necessary and knowledge about reasoning and inferencing needs to be acquired by the software engineers. Up to three

components can be required for integrating data (the integration service, crawler and often the search service), which contribute to the requirements introduced by Semantic Web technologies to the application.

3.2 Mismatch of data models and APIs between components

Within the components of the surveyed Semantic Web applications there were two frequently occurring mismatches from a software engineering perspective: either they internally used different data models or the APIs between components were mismatched, both of which pose important challenges to implementing Semantic Web technologies.

The graph based data model of RDF provides the foundation for knowledge representation on the Semantic Web [1], however programmatically accessing RDF data from a component requires mapping of an RDF graph or subset (in the case of a SPARQL query result) to the data model used by the component [18].

Most of the surveyed applications (92%) were implemented using object oriented languages, and many of the surveyed applications stored data in relational databases. Web applications which are based on relational databases only have to manage the mismatch between object oriented and relational data. Semantic Web applications however have to additionally handle the graph data model of RDF.

Web applications utilise object relational mappers (ORM) such as Hibernate(`www.hibernate.org`) for Java or ActiveRecord(`http://ar.rubyonrails.org/`) for Ruby to transparently map between the data models, and similar approaches for mapping RDF data have been developed such as ActiveRDF [18] for Ruby or SuRF for Python(`http://pypi.python.org/pypi/SuRF`). Without such a mapper, the developer has to provide an abstraction layer on top of the RDF data model himself.

3.3 Missing or belated conventions and standards

In order to benefit from Semantic Web technologies, new paradigms such as the graph based data model of RDF and its open-world semantics need to be understood. On the other hand, many concepts and ideas to which Web application developers are accustomed are hard to translate to the stack of Semantic Web technologies. Approaches for providing conventions and standards to ease the shift towards Semantic Web technologies have often been designed with a considerable delay after the standardisation of RDF in 1999. Providing more and authoritative recommendations is an important factor for increasing adoption of Semantic Web technologies by enterprises.

All of the surveyed applications consume RDF data of some form, 70% allow accessing or importing of user provided external data, and 60% can export data or are reusable as a source for another application. However as discussed in section 3.1, there are many different export and access mechanisms for RDF data, from putting an RDF dump on a web server, embedding links to RDF data in HTML or providing a SPARQL endpoint.

Authoritative recommendations for making RDF accessible over the Web were not available until 2006, when Tim-Berners Lee published a design note(`http://www.w3.org/DesignIssues/LinkedData.html`) which established the Linked Data principles. RDFa specifies how to embed RDF graphs in XHTML documents, and has been in development since 2004. GRDDL (from 2007) specifies how to enable automatic conversion of HTML documents to RDF data.

The basic database interaction pattern of a Web application is "create, read, update and delete"(CRUD) [19], but Semantic Web applications can operate on both local and remote data sources so that updating and deleting depends on the provenance of the data [19]. The survey shows that there is a remarkable difference between the number of surveyed applications which provide a user interface (90%) and the number of applications which allow entering or editing data (30%).

Several approaches for enabling CRUD are currently under development, such as the Update extension for SPARQL, which provides the ability to add, update, and delete RDF data remotely. RDF forms and RDF pushback provide an architecture for structured data input, remote updating of data and conversion of RDF data to legacy data formats. However, at the time of writing the W3C was not involved in these efforts.

3.4 Distribution of application logic across components

The components of a Semantic Web application implement different areas of functionality which are required by Semantic Web technologies, however the components need to be controlled by the application logic in order to use the components for the application domain. For many of the components identified by the survey, the application logic is not expressed as code but as part of queries, rules and formal vocabularies.

58% of the surveyed application use inferencing and reasoning, which often encode some form of domain and application logic, 80% explicitly mention using a formal vocabulary, and 24% 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 Web applications built on top of relational databases [20], and current web frameworks such as Ruby on Rails or the Google Web Toolkit(`http://code.google.com/webtoolkit/`) allow the application developer to control the application interface and the persistent storage of data programmatically through Java or Ruby, without resorting to e.g. JavaScript for the interface and SQL for the data storage. However approaches for centralising the application logic of Semantic Web applications still have to be developed.

3.5 The impact of the challenges on an example application

The challenges of implementing Semantic Web standards become apparent even for small applications. Figure 2 shows the architecture of an application from the authors previous work, the *SIOC explorer* [21]. It aggregates content from weblogs and forums exposing their posts and comments as RDF data using the SIOC vocabulary [12]. The **application logic** and most parts of the application are implemented using the Ruby scripting language and the Ruby on Rails(`http://rubyonrails.org/`) web application framework. The **user interface** allows faceted browsing of the SIOC data and is implemented through the BrowseRDF Ruby component [19]. The **data interface** is provided by ActiveRDF[18], which is an object-oriented Ruby API for accessing RDF data. It is used to access the **integration service**: The data interface is also used to access the **persistent storage** of RDF data using the Redland library(`http://librdf.org/`). Other application data is persistent to a MySQL relational database. The **crawler** is implemented through several Unix command line utilities which are controlled by Ruby. The *SIOC explorer* does not implement a search service or an authoring interface.

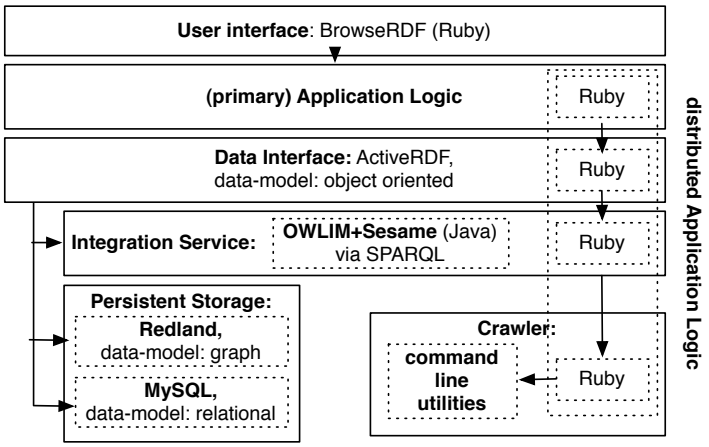

Fig. 2. Result of the architectural and functional analysis of the *SIOC explorer*

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 integration**. 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 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.

4 Mitigating the software engineering challenges

We propose two approaches for mitigating these challenges. The first approach proposes an architectural solution by delegating generic components to external service providers thus simplifying the application. The second approach is to provide better software engineering support with components provided by software frameworks for rapid assembling and customising of complete applications. Both approaches use modularisation to delegate the implementation of some Semantic Web capabilities to components which are provided either by an external service or by a software framework.

4.1 Delegating generic components to external providers

The majority (72%) of surveyed applications implement an integration service, and in section 3.1 we have discussed the issues involved in integrating noisy data even if all sources support RDF. One possible approach to mitigate the identified

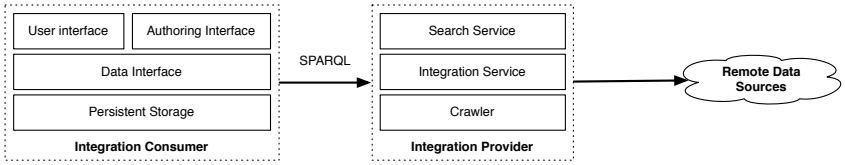

Fig. 3. Simplified Semantic Web application architecture after delegating data discovery, aggregation and integration to an integration provider

challenges is the delegation of generic data discovery, data aggregation and data integration to external providers.

In this way, external **integration providers** can provide the functionality of the integration service, search service and crawler. They provide high value services such as data aggregation and integration, which can be exploited by **integration consumers**. If the cost of discovering, aggregating and integration data on a Web scale is to high for some domains, the integration consumers can benefit from economies of scale when using external integration provider. Figure 3 shows the resulting simplified application architecture. Current search engines already provide APIs which can be utilised to search the whole web or just one specific site, however they lack the capabilities of Semantic Web technologies.

Delegating data integration to external providers can eliminate the need for (1) integration of semantic data if the application only needs generic services, like object consolidation based on inverse functional properties. As new (2) standards and best practices for discovering, publishing and updating of data become available, these only need to be implemented by the integration provider without affecting the **integration consumer**. The (3) mismatch of components is not affected by this, if SPARQL is efficient enough for the application. If domain specific integration of data is required, then different integration providers could provide specialised integration services for individual domains, just like generic and domain specific search engines exist today. However, (4) distributing application logic remains a challenge, as specialised and domain specific queries can contain important parts of the application logic.

The *SIOC explorer* can benefit from this approach by delegating the crawler which aggregates SIOC data from the different weblogs and forums, to an external integration provider such as Sindice [22], which continuously discovers and aggregates SIOC data and performs generic integration services such as object consolidation. This removes two components from the architecture, and allows the application to be purely implemented in Ruby thus eliminating API mismatches. All SIOC data would need to be accessed from Sindice via SPARQL, after which it can be stored in a local persistent store.

4.2 Assembling complete applications from components

While not explicitly described, most surveyed applications are at least partially created on a case-by-case basis: not just the application specific logic is implemented by a software engineer, but also at least one other component of the

application. Very often the *user interface*, *integration service* or the *crawler* are custom made for the specific application. The survey shows that most applications are implemented with more then one programming language, which indicates that most applications are assembled from components with API mismatches.

Software frameworks provide the infrastructure for applications in the form of templates, components and libraries. The provision of software frameworks for implementing Semantic Web technologies has the potential to address all of the identified issues to a certain degree: (1) generic data integration can be provided through libraries provided by the framework. (2) the mismatch of components can be addressed if the framework provides all the components which are necessary to assemble a Semantic Web application, and if all parts of the framework provide APIs for the same programming languages. (3) New standards and best practices, e.g. for data discovery or publication, can be implemented as part of the framework, thus alleviating the need for the application programmer to implement them. (4) Distribution of application logic can be addressed through the framework by providing a central point for implementing the application logic, which can control and customise all of the components.

Similar benefits are already provided by modern Web application frameworks such as Ruby on Rails for Ruby, PHPCake for PHP and Django for Python. The Semantic Web Application Framework (SWAF) [19] provides a first step towards providing components for assembling and customising a complete Semantic Web application.

5 Related work

Our methodology is adapted from [23], which uses six cases studies of software systems as the basis for introducing the basic concepts of software architecture. This is used as the foundation for identifying the most important general challenges for the design of complex software systems which are constructed from many components. We adapt this approach for the field of Semantic Web technologies. The challenges which we identify are based on a survey of 98 Semantic Web applications.

Other empirical surveys about Semantic Web applications are publicly available, however they are not concerned with the software architecture and specific implementation details of concrete applications. Thus they provide no empirical basis for identifying the main challenges of implementing Semantic Web technologies. [24] presents the results of a survey of 627 Semantic Web researchers and practitioners done in January 2007. The questions from the survey cover the categories of demographics, tools, languages and ontologies. It tries to characterise the uptake of Semantic Web technologies and the types of uses cases for which they are deployed. Another similar survey of 161 researchers and 96 application-oriented participants was published online in 2009(`http://preview.tinyurl.com/semweb-company-austria-survey`).

[25] performs a survey and architectural analysis of 35 applications from the "Semantic Web challenges" in 2003, 2004 and 2005. The result is a prescriptive software architecture for Semantic Web applications described with UML. However, the results of the survey do not identify any software engineering challenges for implementing Semantic Web technologies.

While no other empirical analysis of the challenges of implementing Semantic Web applications exist, the ONTOCOM project [7] provides a detailed cost

estimation model for ontology development projects. We do not provide a cost estimate model for software engineering of Semantic Web applications. However, our identification of the main challenges in implementing such applications provides the basis for future research on establishing such cost estimation models.

6 Conclusion

Semantic Web technologies enable new benefits such as semantically structured machine-readable data and the integration of data from multiple, heterogeneous sources. However, adopting new technologies adds effort resulting from implementing the new standards and their associated functionality. We have conducted an empirical survey of Semantic Web applications, which we have used to propose a reference architecture for Semantic Web applications, and for identifying the main challenges which are introduced by implementing Semantic Web technologies: the issues involved in integrating noisy and heterogeneous data, the mismatch of data models and APIs between components, immature and belated best practices and standards, and the distribution of application logic across components. These challenges have been an obstacle for the development of applications exploiting Semantic Web technologies. Two possible approaches for mitigating these challenges are: the simplification of the application architecture by delegating data integration to an external service provider, and assembling and customising of components provided by software frameworks.

The ecosystem of the emerging Web of Data will be based on integration providers and integration consumers. Integration providers provide access to data which has been discovered, aggregated and integrated in a generic way or which caters to a specific domain. Integration consumers will utilise these services to provide their users with benefits which are enabled by the Web of Data and by the integration providers. Data will be published and discovered according to community and industry best practices, which are increasingly implemented by ready-made components. The identified challenges and potential solutions enable future research to better assess the costs of adopting Semantic Web technologies within enterprises, and form the basis for designing better software frameworks and software architecture for exploiting the emerging Web of Data.

Acknowledgements: The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

1. Berners-Lee, T., Hendler, J.A., Lassila, O.: The Semantic Web. Scientific American **284**(5) (2001) 34–43
2. Decker, S., Melnik, S., Van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I.: The semantic web: The roles of XML and RDF. IEEE Internet computing **4**(5) (2000) 63–73
3. Abecker, A., van Elst, L.: Ontologies for knowledge management. In: Handbook on Ontologies in Information Systems. Springer (2004) 453–474
4. Davies, J., Fensel, D., van Harmelen, F., eds.: Towards the Semantic Web: Ontology-driven Knowledge Management. John Wiley and Sons (2002)
5. Fensel, D., Van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An ontology infrastructure for the semantic web. IEEE intelligent systems **16**(2) (2001) 38–45
6. Möller, K.: A Lifecycle Model for Data on the Semantic Web, in progress. PhD thesis, National University of Ireland, Galway (2009)

7. Simperl, E., Popov, I., Burger, T.: ONTOCOM Revisited: Towards Accurate Cost Predictions for Ontology Development Projects. Proceedings of the International Semantic Web Conference (2009)
8. Gerber, A., van der Merwe, A., Barnard, A.: A Functional Semantic Web Architecture. Proceedings of the European Semantic Web Conference (2008)
9. Hausenblas, M.: Building Scalable and Smart Multimedia Applications on the Semantic Web. PhD thesis, Graz University of Technology (2008)
10. Bizer, C., Cyganiak, R., Heath, T.: How to Publish Linked Data on the Web. Technical report, FU Berlin (2007)
11. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering. Pearson Education (2003)
12. Breslin, J., Decker, S., Harth, A., Bojars, U.: Sioc: An approach to connect web-based communities. The International Journal of Web-Based Communities (2006)
13. Hogan, A., Harth, A., Decker, S.: Performing object consolidation on the semantic web data graph. In: Identity, Identifiers, Identification Workshop. (2007)
14. Cyganiak, R., Stenzhorn, H., Delbru, R., Decker, S., Tummarello, G.: Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In: European Semantic Web Conference. (2008)
15. Ding, L., Finin, T.: Characterizing the Semantic Web on the Web. Lecture Notes in Computer Science **4273** (2006) 242
16. d'Aquin, M., Baldassarre, C., Gridinoc, L., Angeletou, S., Sabou, M., Motta, E.: Characterizing Knowledge on the Semantic Web with Watson. In: Workshop on Evaluation of Ontologies and Ontology-based tools of the International. (2007)
17. Kinsella, S., Bojars, U., Harth, A., Breslin, J.G., Decker, S.: An interactive map of semantic web ontology usage. In: Information Visualisation, 2008. IV '08. 12th International Conference. (2008) 179–184
18. Oren, E., Heitmann, B., Decker, S.: ActiveRDF: embedding Semantic Web data into object-oriented languages. Journal of Web Semantics (2008)
19. Oren, E., Haller, A., Hauswirth, M., Heitmann, B., Decker, S., Mesnage, C.: A flexible integration framework for semantic web 2.0 applications. Software, IEEE (2007)
20. Leff, A., Rayfield, J.: Web-application development using the model/view/controller design pattern. Proceedings of the International Enterprise Distributed Object Computing Conference (2001) 118–127
21. Bōjars, U., Heitmann, B., Oren, E.: A Prototype to Explore Content and Context on Social Community Sites. In: Proceedings of the International Conference on Social Semantic Web (CSSW). (2007)
22. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: A document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies (2008)
23. Garlan, D., Shaw, M.: An introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering **1** (1993) 1–40
24. Cardoso, J.: The Semantic Web Vision: Where Are We? IEEE Intelligent Systems **22** (2007) 84–88
25. Cunha, L.M., de Lucena, C.J.P.: Cluster The Semantic Web Challenges Applications: Architecture and Metadata Overview. Technical report, Pontificia Universidade Catolica do Rio de Janeiro (2006)