A Comparative Analysis of Linked Data Tools to Support Architectural Knowledge

Cristina Roda *cristinarodasanchez@gmail*
University of Castilla - La Mancha
Albacete, Spain

Elena Navarro *elena.navarro@uclm.es*
University of Castilla - La Mancha
Albacete, Spain

Carlos E. Cuesta *carlos.cuesta@urjc.es*
Rey Juan Carlos University
Móstoles, Madrid, Spain

Abstract

Architectural Knowledge (AK) has been an integral part of Software Architecture specification since its original inception, but it has not been explicitly managed until recently. It can be described as a computational structure composed of design decisions and rationales. Recent research emphasizes that availability must be complemented by an effective use of this information. We propose the use of Linked Data techniques to define and manage AK, thus achieving flexible storage and scalable search. Our approach suggests storing the network of decisions in RDF format to be retrieved efficiently by means of SPARQL queries. As a side effect, many different AK structures can be described in this way, which then becomes a general format to describe AK. Using this approach, this work analyses some significant features regarding AK of several Linked Data tools, in order to determine which ones are the best/worst for sharing and reusing AK as Linked Data.
Keywords: Architectural Knowledge, Linked Data, Ontology, RDF, SPARQL, AK Network.

1. Introduction

In recent times, *Architectural Knowledge* (AK) has become one of the most popular topics in the Software Architecture (SA) field. AK has an organizational perspective, but it can also be described as a computational structure, composed of assets of design knowledge, tracing back to specific requirements and forward towards an implementation: that is, it describes the extended history of the system's design [10]. Then, from this perspective, AK is composed of architectural elements, requirements, and a number of design assets. When the specification just provided the final architecture, all this information was *unrepresented design knowledge* [21]; but now the architecture includes this information as part of the *rationale*.

Although AK was considered as an integral element of the SA specification since the very beginning of the discipline [14], it has been only recently [3] that both researchers and practitioners have become aware of its importance. The importance of AK goes beyond merely documenting the architecture; it is one of the vehicles for ensuring the quality of the software development process. For instance, Bratthall et al. [4] carried out a survey with several subjects from both industry and academia. They concluded that most of the interviewed architects considered that by using the AK they could shorten the time required to perform the change-tasks. Interviewed subjects also concluded that the quality of the results, when they had to predict changes on unknown real-time systems, was better using AK. Ozkaya et al. have also concluded, during their interview study [13], that the difficulties during both the initial phases and the evolution of systems are not only due to the availability

of AK, but depend on its effective use. However, despite these difficulties, interviewed architects have also remarked that they perceive AK to be essential when evolution happens. Feilkas et al. [6] have also carried out a case study on three industrial information systems highlighting the relevance of AK as one of its research questions. Authors have detected that one of the problems in these projects was that developers were not aware of the intended architecture because the AK was not properly described. Moreover, the case study also exposed that both a machine-readable form of the AK and the introduction of automatic analysis techniques are keys to achieve architecture-awareness in the development team.

We believe that a relevant conclusion can be extracted: it is a *must* to facilitate the management of AK, not only during the development process, but especially during the evolution of systems, in order to provide them with certain quality levels. The proposal presented in this work paves the way to satisfy this need by using a *Linked Data* (LD) [2] approach. In general, Linked Data [2] is referred to use the Web to create typed links between data from different sources. In this manner, the main purpose of our approach is to provide architects with abilities for both documenting the AK and exploiting it, regardless of the architecture-centric practices already used by their organization.

Therefore, our initial hypothesis is that LD provides a good basis for software architects to describe and manage the AK, and that this technology provides an adequate toolset which can be exploited by them in a flexible way. In this line of research, it is followed the *design science research* method [7] that involves the design of novel artifacts whose use and performance let improve/understand the behavior of Information Systems. In this work, we have selected some LD tools which seem to be particularly relevant, and examine their features from an AK perspective. Our purpose is to decide (a) whether these features seem to be adequate enough for supporting AK, thereby justifying this approach; and (b) which one of these tools, if any, seems to be more promising from an AK perspective.

Our approach tries to contribute to the emerging *Organizational Social Structures* (OSS) [18] perspective. An OSS can be seen as a dynamic interplay of people, e.g. stakeholders or developers. From this perspective, we can find two alternatives: *Communities of Practice* (CoP), where people share/learn a common practice, and *Strategic Communities* (SCs), where people share experience and knowledge with the aim of achieving strategic business advantage. Both of them have a common need: the need of sharing knowledge to achieve their goals. This work intends to establish a foundation for them by means of the LD approach.

The remainder of this paper is organized as follows. Section 2 outlines some Linked Data tools that are able to manage AK as Linked Data. All of these tools are afterwards analysed in Section 3, according to different features, in order to find out which ones seem to be the best/worst for sharing and reusing AK in a Linked Data format. Finally, Section 4 presents our conclusions and future work.

2. Linked Data Tools

This section presents some of the most widely used Linked Data tools (a more detailed description about their features, installation, etc., can be found in [16]). Initially, they have been selected with the following requirements: they can handle LD and they must have a wide support. Furthermore, all these tools have been selected because they have support for almost all the features that we have considered in the analysis presented in Section 3. Some of these features are of vital importance to manage AK, such as the SPARQL query language, RDF and OWL schemas, RDF input data format, and RDF/XML as serialization format. In particular, these four features are compulsory in our analysis so that those tools that do not support them will be penalized regarding to their total score.

2.1 Virtuoso

It is a single data server, developed by *OpenLink Software*, that offers functionality for both a traditional Relational data management server and a Linked Data server [12]. Namely, it provides RDF, Relational and XML data management, document and Linked Data server, as

well as a web application server, among others. In order to use this server with Linked Data, we may use *OpenLink Data Spaces* (ODS) [11][12] that allows one to establish and manage data on the web, as an extension of the emerging Semantic Web. It includes the platform *ODS Briefcase* which enables users to control file access rights, content-based search and metadata. In addition, all resources are exposed as RDF data sets, so that the file server functionality can be exploited by means of the SPARQL query language for the Semantic Web. ODS Briefcase offers several features especially suited for managing LD, such as (i) uploading RDF files and validating their format according to a particular syntax, e.g. XML; (ii) editing these files; (iii) consuming data uploaded to the server by means of SPARQL [11]; and (iv) showing query results in different formats (HTML, XML, JSON, Javascript, NTriples, RDF/XML or spreadsheet). In addition, it is able to define SPARQL queries using a specific graphical representation, and to run them using *OpenLink iSPARQL* [11].

2.2 Linked Media Framework (LMF)

LMF [17] is an easy-to-setup application server which packages Semantic Web technologies to offer advanced services. This framework consists of two main elements: *LMF Core* and *LMF Modules*. The LMF Core component is a LD server that exposes data following LD Principles. In addition, it also offers a highly configurable Semantic Search service and a SPARQL endpoint. Moreover, some other elements which can be used are [17]: (i) *LMF Semantic Search*, that offers a highly configurable Semantic Search service; (ii) *LMF Linked Data Cache,* which implements a cache to the LD Cloud, to be used transparently when querying the contents of the LMF. In the case of querying a local resource that links to a remote resource in the Linked Data structure, the remote resource will be retrieved in the background and cached locally.

2.3 Apache Jena & Fuseki

Apache Jena [19] is a Java framework for building Semantic Web applications, and specially *Jena* is a Java API for these kind of applications that can be used to create and manipulate RDF graphs. Jena provides a collection of tools and Java libraries to develop semantic web and LD applications, tools and servers. Namely, Jena includes an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats. It has also an ontology API for handling OWL and RDFS ontologies and a rule-based inference engine for reasoning with RDF and OWL data sources. It is able to efficiently store large numbers of RDF triples on disk. It has a query engine compliant with the latest SPARQL specification and a server to allow these RDF data to be published so that they can be used by other applications using a variety of protocols. In addition, it provides constant classes for well-known schemas (RDF, RDFS, RDFa, Dublin Core or OWL) and also has some methods for reading and writing RDF as XML. On the other hand, *Fuseki* is a SPARQL server that offers services for SPARQL update and file upload to a selected dataset, validators for SPARQL query and update, and for non-RDF/XML formats.

2.4 TopBraid Suite

TopBraid Suite [20] offers semantic technology applicable in several scopes, such as to connect data, systems and infrastructures or to build flexible applications from LD models. All components of the suite work within an open architecture platform built specifically to implement W3C standards for the integration and combination of data obtained from diverse sources. These components are *TopBraid Composer*, an Eclipse plug-in that provides complete support for developing and managing ontologies and LD; *TopBraid Ensemble* [20], a semantic web application assembly toolkit for rapid configuration and delivery of dynamic business applications, suitable to create model-driven applications; and *TopBraid Live* (TBL), a server to deploy flexible, model-driven applications and dynamic, on-demand integration of data from diverse sources.

2.5 Sesame

Sesame is an open source Java framework for storing and querying RDF data, similar to Jena (Section 2.3). This framework is fully extensible and configurable with respect to storage mechanisms, inference engines, RDF file formats, query result formats and query languages. The core of the Sesame framework is the *RDF Model API,* which defines how the building blocks of RDF (statements, URIs, blank nodes, literals, graphs and models) are represented. Sesame also provides the *Repository API,* which describes a central access point for Sesame repositories. Its purpose is to give a developer-friendly access point to RDF repositories, offering various methods for querying and updating the data in an easy way. Additionally, Sesame supports the use of SPARQL for querying memory-based and disk-based RDF stores, RDF schema inference engines, as well as explicit support for the most popular RDF file formats and query result formats.

2.6 Mulgara

Mulgara [9] is a scalable open source RDF datastore written in Java, under the Open Software License 3.0. This tool can be considered akin to a relational database, as the information can be stored and retrieved via a query language. But unlike a relational database, Mulgara is optimized for the storage and retrieval of RDF statements, i.e. subject-predicate-object. Some of its main features are native RDF support, multiple databases per server, a simple SQL-like query language (similar to SPARQL), large storage capacity or low memory requirements. Moreover, Mulgara provides mechanisms for ensuring *reliability* (full transaction support, clustering and store level fail-over, permanent integrity), *connectivity* (using SOAP, Jena, etc.), *manageability* (near zero administration, web based configuration tools) and *scalability* (via XA Triplestore engine) of our system.

2.7 RedStore

RedStore [8] is a lightweight RDF triplestore written in C, which uses the Redland library, a set of free software libraries that provides support for RDF. It supports, in addition to native persistence and in-memory storage, a variety of storage backend adapters, including MySQL, Postgres and Virtuoso. In native mode, RedStore uses hashtables to store RDF data. Its main features are: SPARQL over HTTP support, a built-in HTTP server, support for a wide range of RDF formats, and a test suite for unit and integration testing.

2.8 Callimachus

Callimachus [1] is a framework for data-driven applications based on LD. It enables web developers to quickly create web applications based on LD, as they only need a web browser to develop a data-driven application. In addition, Callimachus uses either Sesame (Section 2.5) or Mulgara (Section 2.6) for RDF storage, AliBaba (a RESTful object-RDF library), and a proprietary template-by-example technique to view and edit resources. One of the most interesting features of Callimachus is that it is able to execute queries using RDF itself.

3. Feature Analysis

This section presents a detailed analysis of the LD tools from the previous section in order to compare their features and determine what the best ones are. This analysis was performed using the examples of AK networks described in *http://goo.gl/NJD2Ft* to illustrate the power of LD. Some of these features have been chosen because we consider them of vital importance for solving the following deficiencies which often make even more difficult the exploitation of AK:

i. There is not a standard for representing AK, but multiple approaches.
ii. Not every decision and rationale throughout the lifecycle (the history of AK) is recorded.

iii. There is not a standard language for querying AK.
iv. There is not a scalable solution able to manipulate the historical AK.

Therefore, the features selected from the *point of view of managing AK as LD* are:

- *Data persistence:* indicates if the LD tool provides persistence for its stored data. It is related to deficiency (ii), as this feature is desirable to access the history of AK over time.
- *Query languages:* identifies the different query languages that can be used to manipulate LD. It is related to deficiencies (iii) and (iv), as we want to store AK and the volume of such information can rapidly increase, it is also desirable to have a query language that allows us to navigate AK efficiently.
- *Supported schemas/vocabularies* by the LD tool, like XML or RDF. It is related to deficiency (i), as we are looking for a standard schema to represent AK, such as RDF.
- *Federated queries:* indicates if the LD tool supports data searching across multiple servers. It is related to deficiencies (iii) and (iv), as AK may be stored in different locations, following a LD approach.
- *Input data formats* supported by the LD tool in order to store and manage LD. It is related to deficiency (i), so that the standard for storing AK could be RDF.
- *Query output formats* provided by the LD tool when a query is executed. It is related to deficiency (iii), as we are looking for a standard query language that provides different output formats when querying AK.
- *RDF serialization formats* supported by the LD tool, like RDF/XML, Turtle, etc. It is related to deficiency (i), as we want to represent our AK with a standard such as RDF.

On the other hand, we also take into account some technical features that are relevant when choosing a software tool. They are not directly related to the AK field and its management as LD, but to the software itself. In this way, features selected from the *point of view of software* are:

- *Type of tool:* specifies the type of the analyzed LD tool.
- *Interaction UI* (User Interface): indicates whether the LD tool has a friendly UI.
- *License:* informs of the type of license that the LD tool has.
- *Security:* indicates how the LD tool guarantees that the information is always safely stored.
- *SDK:* indicates if the LD tool can be used to create other applications.
- *Complexity*: how complex is the *Installation*, *Start-up* and *Data management* of the tool.

A detailed analysis is presented in [16], providing a thorough analysis of all these features (both from the point of view of managing AK as LD and from the point of view of software) with regard to each tool. Notice that almost all LD tools are able to provide federated queries, as they all support SPARQL 1.1 [22]. This language can be used to express queries across diverse data sources, regardless of whether the data are stored natively as RDF or retrieved as RDF via middleware. *SPARQL 1.1 Federated Query extension* has been created to execute queries distributed over different SPARQL endpoints.

In order to decide which the best/worst LD tools are with regard to the analysed features, the following formula (which maximizes the global contribution) is used:

Total score = QL1 · SS1 · IDF1 · RSF1 · (IUI + DP + QL2 + SS2 + FQ + IDF2 + QOF + Li + Sec + RSF2 + SDKS + CI + CSU + CDM)

Each one of these parameters is calculated using the rules presented in Table 1, namely in column *How to score*. Parameters QL1, SS1, IDF1 and RSF1 are compulsory features, and therefore their values will be either 1 or 0, depending on their support. The reasons to consider them as necessary features are the following:

- QL1 establishes that the tool supports at least SPARQL as a query language, due to the fact that it is the standard query language used in LD.
- SS1 determines if the tool supports RDF, because this is the standard representation schema accepted for LD. It also determines that the tool supports OWL, which is widely

used for reasoning in the context of LD, because it enables to connect different datasets, located in different data stores, using different schemas. Moreover, both RDF and OWL can help architects to obtain more complete answers for queries over LD, as stated in [15].

- IDF1 establishes that the tool provides RDF as input data format, as it is the standard input data format for LD.
- RSF1 indicates that the tool provides RDF/XML as a RDF serialization format, as it has been defined by the W3C in the original specification.

Initially, we did not consider these four compulsory parameters within the formula, but only the sum of the remaining ones. Finally, we noticed that this was a wrong choice, given that some tools obtained better scores than others, despite they lacked an adequate support for LD, e.g. lack of schemas for RDF and OWL. This led us to mark them as compulsory. This way, the tool with the highest score must be considered as the best analysed LD tool and conversely, the one with the lowest score will be considered the worst. Notice that the remaining features are still considered equally important; therefore they are equally scored, namely *1 point* per each feature at most –so the score of each feature ranges between 0 and 1. Moreover, in order to facilitate this normalization, some of these features, such as QL2 or SS2, are calculated by dividing the number of supported query languages or schemas by the number of query languages or schemas offered by the tool with the highest support.

Table 2 shows the marks for each LD tool. Notice that we have omitted the feature that is

Table 1. Features preferences.

Analysed feature		Parameter	Preferences	How to score
Type of tool		-	(Without preferences)	(Not scoring, only informative)
Interaction UI		IUI	A User Interface that allows users to interact with the Linked Data tool.	**1** If the tool has a UI **0** Otherwise
Data persistence		DP	A platform that provides data persistence over the time.	**1** If the tool provides data persistence **0** Otherwise
Query languages*		QL1	A tool that supports, at least, SPARQL.	**1** If the tool supports SPARQL **0** Otherwise
		QL2	The more query languages supported, the more preferable.	(Number of supported query languages)/5
Supported schemas/vocabularies*		SS1	A tool that supports, at least, RDF and OWL as schemas.	**1** If the tool supports RDF and OWL as schemas **0** Otherwise
		SS2	The more schemas are supported, the more preferable.	(Number of supported schemas)/13
Federated queries		FQ	A tool that supports federated queries.	**1** If the tool provides federated queries **0** Otherwise
Input data formats*		IDF1	A tool that supports, at least, RDF as an input data format.	**1** If the tool supports RDF as an input data format **0** Otherwise
		IDF2	The more input data formats supported, the more preferable.	(Number of input data formats)/11
Query output formats		QOF	The more query output formats supported, the more preferable.	(Number of query output formats)/17
License		Li	A tool with an Open Software license.	**1** If the tool has an Open Software license **0** Otherwise
Security		Sec	A tool with security services.	**1** If the tool has security services **0** Otherwise
RDF serialization formats*		RSF1	A tool that supports, at least, RDF/XML as a RDF serialization format.	**1** If the tool supports RDF/XML as a RDF serialization format **0** Otherwise
		RSF2	The more RDF serialization formats are supported, the more preferable.	(Number of RDF serialization formats)/10
SDK support		SDKS	A tool that provides SDK support.	**1** If the tool has SDK support **0** Otherwise
Complexity	Installation	CI	It is desirable a Low complexity in all cases.	**0.2** Low **0.1** Medium **0** High
	Start-up	CSU		**0.3** Low **0.15** Medium **0** High
	Data management	CDM		**0.5** Low **0.25** Medium **0** High

(*) These features are compulsory.

Table 2. Scores of Linked Data tools.

Analysed features	Par.	Virtuoso	LMF	Apache Jena & Fuseki	TopBraid Suite	Sesame	Mulgara	RedStore	Callimachus
Interaction UI	IUI	1	1	1	1	1	1	1	1
Data persistence	DP	1	1	1	1	1	1	1	1
Query languages	QL1 QL2	1 5/5=1	1 1/5=0.2	1 2/5=0.4	1 1/5=0.2	1 2/5=0.4	1 2/5=0.4	1 3/5=0.6	1 1/5=0.2
Supported schemas/vocabularies	SS1 SS2	1 4/13=0.31	0 4/13=0.31	1 9/13=0.69	1 5/13=0.39	1 12/13=0.92	1 4/13=0.31	0 1/13=0.08	1 13/13=1
Federated queries	FQ	1	1	1	1	1	0	1	1
Input data formats	IDF1 IDF2	1 4/11=0.36	1 4/11=0.36	1 2/11=0.18	1 11/11=1	1 1/11=0.09	1 4/11=0.36	1 1/11=0.09	1 1/11=0.09
Query output formats	QOF	7/17=0.41	5/17=0.29	6/17=0.35	4/17=0.24	5/17=0.29	1/17=0.06	17/17=1	1/17=0.06
License	Li	1	1	1	0	1	1	1	1
Security	Sec	1	1	1	1	1	0	1	1
RDF serialization formats	RSF1 RSF2	1 5/10=0.5	1 5/10=0.5	1 8/10=0.8	1 4/10=0.4	1 9/10=0.9	1 3/10=0.3	1 10/10=1	1 3/10=0.3
SDK support	SDKS	0	0	1	0	1	0	0	0
	CI	0.2	0.2	0	0.2	0.1	0.2	0	0.1
	CSU	0.3	0.3	0.15	0.3	0.15	0.3	0.15	0.3
	CDM	0.5	0.25	0.25	0.25	0.5	0.5	0.5	0.25
TOTAL SCORE		*8.58*	*0*	***8.82***	*6.98*	***9.35***	*5.43*	*0*	*7.3*

not scoring, i.e. *Type of tool*. As a result, the first position is for *Sesame* (9.35), the second is for *Apache Jena and Fuseki* (8.82), then *Virtuoso* (8.58), *Callimachus* (7.3), *TopBraid Suite* (6.98) and *Mulgara* (5.43). The last ones are *RedStore* (0) and *Linked Media Framework* (0), which have no score because they do not support OWL (a compulsory feature represented as parameter SS1). As we can see, the best LD tools, between the analysed ones and according to our ranking, are *Sesame* and *Apache Jena & Fuseki*. In this sense, these two tools are really similar, given that both provide a Java framework to manage LD. Their strength with respect to the others comes from the large number of supported schemas and vocabularies and RDF serialization formats, in addition to the SDK support, which is exclusive of them.

4. Conclusions and Future Work

As already indicated, AK provides the basis to guide architects in many decisive processes, such as evolution [5], and to achieve certain levels of quality during these processes. However, as mentioned in Section 3, there are several shortcomings that can prevent organizations from exploiting AK, such as the variety of formats for documenting and representing it, or the difficulties in meeting its requirements for evolution.

In this paper, we suggest the use of LD techniques to solve these shortcomings. These techniques allow one to define and query AK in an easy and effective way, providing a flexible storage and scalable search. In this sense, our approach recommends storing the network of decisions using RDF, to be efficiently retrieved by means of SPARQL queries. It is worth noting that this proposal does not depend on any particular AK tool or model, or on any RDF triplestore or serialization format: we are considering the benefits of the LD approach itself. Furthermore, we have presented several tools for managing LD and we have analysed some relevant features of these tools with regard to AK, with the purpose of deciding to which extent they provide relevant features, and which ones are the best/worst for sharing and reusing AK as LD. Our results show that the best LD tools for this specific goal, among the analysed ones, are *Sesame* and *Apache Jena & Fuseki*. Therefore, given these outcomes, we conclude that the best choice to handle and share AK is to use a LD tool with some SDK support, such as Sesame or Apache Jena & Fuseki, to build some specific tool or interface which simplifies the use of this tool in an AK context. In summary, the LD approach offers important advantages in terms of scalability, as it has been defined to manage great amounts of data thanks to its index-based structure. This facility is being widely used within the open data initiative, to make data available to everyone. This approach makes possible to define and implement a shared repository of AK available to every architect, without compromising specific approaches.

Several ideas outline the path for our future work, due to the possibilities offered by LD. One of them is related to the social part of the proposal. Even though OSS are already gaining a growing attention, and practitioners are becoming more aware of the need of sharing the knowledge, we are planning to carry out several case studies in order to evaluate the degree of acceptance of the ideas presented in this work.

Acknowledgements

This work has been funded in part by the Spanish Ministry of Economy and Competiveness under the National R&D&I Program, within Projects DESACO (TIN2008-06596-C02-01), and CoMobility (TIN2012-31104), and also through the FPU scholarship (FPU12/04962).

References

1. 3 Round Stones, I.: Callimachus, http://callimachusproject.org/.
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. J. Semant. Web Inf. Syst. 5 (3), 1–22 (2009).
3. Bosch, J.: Software Architecture: The Next Step. 1st European Workshop in Software Architecture (EWSA'04), pp. 194–199. Springer (2004).
4. Bratthall, L., Johansson, E., Regnell, B.: Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software. 2nd Int. Conf. on Product Focused Soft. Process Improvement (PROFES 2000). pp. 126–139, Springer (2000).
5. Cuesta, C.E., Navarro, E., Perry, D.E., Roda, C.: Evolution styles: using architectural knowledge as an evolution driver. J. Softw. Evol. Process 25 (9), 957–980 (2013).
6. Feilkas, M., Ratiu, D., Jurgens, E.: The loss of architectural knowledge during system evolution: An industrial case study. 17th IEEE International Conference on Program Comprehension (ICPC'09). pp. 188–197, IEEE CS Press (2009).
7. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Q. 28 (1), 75–105 (2004).
8. Humfrey, N.J.: RedStore, http://www.aelius.com/njh/redstore/.
9. Mulgara Project: Mulgara, http://www.mulgara.org/.
10. Navarro, E., Cuesta, C.E.: Automating the Trace of Architectural Design Decisions and Rationales Using a MDD Approach. 2nd European Conf. on Software Architecture (ECSA 2008). pp. 114–130, Springer (2008).
11. OpenLink Software: OpenLink Virtuoso, http://my.openlinksw.com.
12. OpenLink Software: Virtuoso Universal Server, http://virtuoso.openlinksw.com/.
13. Ozkaya, I., Wallin, P., Axelsson, J.: Architecture knowledge management during system evolution. 2010 ICSE SHARK '10. pp. 52–59, ACM Press (2010).
14. Perry, D.E., Wolf, A.L.: Foundations for the Study of Software Architecture. ACM Softw. Eng. Notes 17 (4), 40–52 (1992).
15. Polleres, A., Hogan, A., Delbru, R., Umbrich, J.: RDFS & OWL Reasoning for Linked Data. Reasoning Web. Semantic Technologies for Intelligent Data Access. pp. 91–149, Springer (2013).
16. Roda, C., Navarro, E., Cuesta, C.E.: Analyzing Linked Data tools for SHARK. TR#DIAB-13-09-1. Computing Systems Department, Albacete, Spain (2013).
17. Semantic Web: Linked Media Framework, http://semanticweb.org/wiki/Linked_Media_Framework.
18. Tamburri, D.A., Lago, P., Van Vliet, H.: Organizational Social Structures for Software Engineering. ACM Comput. Surv. 1–34 (2012).
19. The Apache Software Foundation: Apache Jena, http://jena.apache.org/.
20. TopQuadrant Inc.: TopQuadrant, http://www.topquadrant.com.
21. Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Softw. 22 (2), 19–27 (2005).
22. W3C: World Wide Web Consortium, http://www.w3.org/.