Formal ontologies and data shapes within the Software Engineering development lifecycle (TSE)

Jose María Alvarez Rodríguez
Department of Computer Science and Engineering
Carlos III University of Madrid
Madrid, Spain
josemaria.alvarez@uc3m.es

Valentín Moreno, Juan Llorens
Department of Computer Science and Engineering
Carlos III University of Madrid
Madrid, Spain
{vmpelayo,llorens}@inf.uc3m.es

Abstract— **Business models, organizational activities and corporate strategies are now being reshaped to meet the new needs of a challenging and evolving environment in which more up-to-date, secure, safer, cost-efficient and personalized software products and services must be timely delivered. This new digital context also represents an opportunity for the improvement and extension of existing software engineering methods. One of the current trends in Software Engineering development lies in boosting interoperability and collaboration between tools and people through the sharing of existing artifacts under common data models, formats and protocols to improve the practice and reuse of existing software artifacts. In this context, formal ontologies and data shapes play a key role to model and exchange data and to provide services for data validation (consistency checking) or type inference as part of a knowledge management strategy. In this document, an initial review of the different approaches to model and exchange data of software artifacts is done to finally evaluate and discuss the proper mechanisms to technically support the upcoming needs in the Software Engineering development lifecycle.**

Keywords: software development lifecycle; ontologies; data shapes; interoperability; knowledge representation

I. Introduction

The Digital Age, the "Society 5.0" in Japan or the "4th Industrial Revolution" [1] has come to say that any industry, business or even our daily life will suffer a deep transformation being driven by software systems.

This new digital context also represents a challenge and an opportunity for the improvement and extension of existing software engineering methods. After 50 years [2], the Software Engineering (SE) discipline has focused on ensuring the efficiency, correctness, robustness and reliability of software systems. The development of some SE knowledge areas [3] have set the foundations of the discipline. Furthermore, the definition of methodologies, methods and models, software development lifecycles and their technological support have also represented a major step to improve the SE practice from both organizational and scientific/technological points of view[4] [5].

On the other hand, knowledge management techniques have gained enough momentum in the SE discipline to elevate the meaning of the implicit knowledge coded into software pieces. Software as a knowledge asset is becoming a commodity that is embedded in products, business or manufacturing processes. It is a new kind of intellectual asset that can help us to reduce costs and time to market, and to generate competitive advantage. In this light, knowledge management techniques [6] can be applied to capture, structure, store and disseminate software artifacts to directly support methods such as software reuse.

However, one of the cornerstones in knowledge management lies in the selection of an adequate representation paradigm. After a long time [7], this problem still persists since a suitable representation format (and syntax) can be reached in several ways.

Obviously, different types of knowledge, such as those in the SE discipline (requirements, source code, test cases, etc.), require different types of representation [8] [9]. But, on the other hand, knowledge management also implies the standardization of data and information within the development lifecycle. Any bit of information must be structured and stored for supporting other application services such as business analytics or knowledge discovery.

One of the current trends in SE development lies in boosting interoperability and collaboration between tools and people through the sharing of existing artifacts under common data models, formats and protocols to improve the practice and reuse of existing software artifacts. In this context, OSLC ("Open Services for Lifecycle Collaboration"), model-driven approaches, etc. are defining a collaborative software development ecosystem [10] through the definition of data shapes that serve us as a contract to get access to information resources through standardized.

In particular, the Representational State Transfer (REST) software architecture style is commonly used to manage information and software resources such as requirements, test cases or even source code that are publicly represented and exchanged in standard formats such as RDF or just XML. Obviously, these approaches represent a big step towards the integration and interoperability between the agents involved in the software development lifecycle.

However, a knowledge management strategy to take advantage of software artifacts is beyond of the mere exchange of data. Consistency checking, type inference, data integrity, etc. are common operations expected in a knowledge-centric framework that can help us to improve the practice in the SE discipline.

DOI reference number: 10.18293/SEKE2019-114

In this paper, authors review and discuss the role of ontologies and data shapes as a mean to represent, exchange and consume software-related artifacts with the aim of improving and enriching software development methods providing capabilities for consistency checking, type inference or data integrity.

II. Background

The Software Engineering discipline [11] [12] [13] initially focused on the definition of methodologies and methods to tackle the problem of reliability in software systems and to ease the reuse of working software products.

The notion of software engineering process was then defined, and different software development lifecycles (SDLC) such as the Structured Systems Analysis and Design Method (SSADM), the Waterfall model, the Rational Unified Process (RUP), the Rapid application development (RAD) or the Vee model were established as a method to manage the complexity of software development. Afterwards, new programming paradigms, standard notations (e.g. UML or OCL), languages and tools emerged as way of improving the abstract thinking in combination with a technological support providing better development environments.

Once, the methods and the supporting tools were available, the focus was the reuse of software assets, the quality management, the definition of maturity models and the automation of tasks generated during the software development process. More specifically, the software reuse area gained momentum through the definition of methods to reuse existing software components through the abstract description of different elements, e.g. architectural and design patterns, libraries, component models or services. Furthermore, coding practices [14] were also improved adding new foundations such as the SOLID[1] principles or new practices such as refactoring.

Initiatives such as Model-Driven Engineering [15] [16] (MDE) and Model-Driven Architecture (MDA) later posed the foundations to automate the production of software for specific domains. In the last decade, software product lines [17] have also been subject of study and application as a method for automating the creation of families of software products.

At the development level, the emerging use of Agile methods [18], inspired by the Spiral model, such as XP, Scrum or Kanban and, processes such as BDD (Behavior-Driven Development) or TDD (Test Driven-Development) have also led us to improve the software development practice in combination with approaches such as DevOps [19] (Development and Operations). The latter are often applying to ease the continuous transition of software systems from a development to a production environment as a mean for the early detection of "bugs" and the improvement of the maintenance and change management processes.

In summary, the Software Engineering discipline has reached a great maturity level. There is a huge body of knowledge and practice that allow us to acknowledge which are, and which are not, the software engineering practices that really represent timeless scientific and technological foundations for a proper software development process. That is why, knowledge management techniques are aimed at enhancing existing SE methods by providing a knowledge layer on top of the data generated during the development lifecycle.

III. Formal Ontologies vs Data Shapes

In the early days of the Semantic Web, formal ontologies [20] designed in RDFS (Resource Description Framework Schema) or OWL were the key technologies to model and share knowledge.

From upper ontologies such as DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) or SUMO (The Suggested Upper Merged Ontology) to specific vocabularies such FOAF (Friend Of A Friend) or SKOS (Simple Knowledge Organization System), the process to share knowledge consisted in designing a formal ontology for a particular domain and populate data (instances) for that domain. Although the complete reuse of existing ontologies was expected, the reality demonstrated that every party willing to share knowledge and data would create its own ontologies. Thus, the main idea behind web ontologies was partially broken since just a few concepts were really reused.

Once the Linked Data initiative (RDF + HTTP) emerged to unleash the power of existing databases, a huge part of the Semantic Web community realized that a formal ontology was not completely necessary to exchange data and knowledge.

Taking into account that ontologies were still present, the efforts were focused on providing methods for data consistency [21] through the execution of procedures such as: 1) reasoning processes to check consistency, and 2) rules, mainly in SWRL (Semantic Web Rule Language) or SPARQL [22] [23]. These procedures are not fully recommended, due to performance issues, when a huge number of instances are available. As a new evolution, then, the community realized that ontology-based reasoning was not the most appropriate method for data validation when data is being exchanged.

That is why in recent times the Semantic Web community has seen an emerging interest to manage and validate RDF datasets according to different shapes and schemes. A data shape can be defined as a resource, i.e. metamodel, that describes contents of, and constraints on, other resources. In this way, it is possible to not just improve but to overcome some of the well-known restrictions [24] of RDF encoded data: 1) lack of support to represent certain knowledge features N-ary relationships [25], 2) practical issues dealing with reification [26] and blank nodes [27]. New specifications and methods for data validation (consistency) are being designed to turn reasoning-based validation into a kind of grammar-based validation.

[1] "The Principles of OOD". Source: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod (last visited: 20th of February, 2019)

Process	Type	Creation	Scope	Refs.
Consistency check	Vocabulary-based	Semantic Web reasoner	RDF datasets	[21]
Data validation (integrity)	Query-based	Hand-made RDF templates	RDF datasets	[22]
	Vocabulary-based	Hand-made	RDF datasets	[23]
	Vocabulary-based	Hand-made or automatically generated by an OSLC API	OSLC Resource Shape	[28]
	Vocabulary-based	Hand-made	Dublin Core Description Set Profiles	[29]
	Query-based	RDF Unit (test creation)	RDF datasets	[30]
	Query-based (generated from ShEX expressions)	Automatic generation of SPARQL queries	RDF datasets	[31] [32] [33]
	Vocabulary and Query-based	Automatic generation of SPARQL queries	OWL and RDF under Closed World Assumption	[34]
	Query-based	SPIN language + SPARQL queries	RDF datasets	[35]

Table 1 Comparison of methods for RDF data validation.

These methods take inspiration from existing approaches in other contexts such as DTD (Document Type Definition), XML-Schema or Relax NG (REgular LAnguage for XML Next Generation) for XML, or DDL (Data Definition Language) for SQL (Structured Query Language).

The W3C launched in 2014 the RDF Data Shapes Working group having as main outcome the SHACL (Shapes Constraint Language), W3C Recommendation, and the ShEX (Shape Expressions) language [31] [32]. Both are formal languages for expressing constraints on RDF graphs including cardinality constraints as well as logical connectives for disjunction and polymorphism. OSLC Resource Shapes [28], Dublin Core Description Set Profiles [29], and RDF Unit [30] were also other constraint languages for data validation in the context of Linked Data.

Following a more classical approach for RDF data validation, the Pellet Integrity Constraints is an extension of the existing semantic web reasoner [34] that interprets OWL ontologies under the Closed World Assumption with the aim of detecting constraint violations in RDF data. These restrictions are also automatically translated into SPARQL queries. This approach has been implemented on top of the Stardog[2] database, enabling users to write constraints in SPARQL, SWRL or as OWL axioms. This approach has been reported as a method for data validation and type inference in some case studies (e.g. NASA Knowledge Graph)[3] were logical models are translated into OWL instances and, then, a reasoning process (semantic web based or rule-based) is executed to find inconsistencies, infer types, etc. However, the complexity and time to make transformations between object models and Description Logics seems to be not very effective since the same information is being modelled at the same time under two different paradigms. Finally, the SPIN language [35] also makes use of SPARQL (mainly its syntax) to define constraints on RDF-based data that can be executed by systems supporting SPIN (SPARQL Inferencing Notation), such as the TopBraid's toolchain.

In conclusion, the relevance of data validation to exchange RDF-encoded data is clear. RDF Data Shapes in its different flavors, such as OSLC Resource Shapes, are becoming the cornerstone for boosting interoperability among agents, see Table 1. It is also clear that ontologies are becoming less important although a combined approach (data shapes and a formal ontology) can provide important benefits in terms of data validation and knowledge inference (if needed). In the context of SE and reuse, as it has been previously outlined, software artifacts must take advantage of new technologies to enable practitioners the automatic processing of exchanged data.

IV. Knowledge Management within the Software Engineering process

Software is becoming a commodity, knowledge that is embedded in every product, business and development or manufacturing process. Assuming that a software artifact is a knowledge asset, the ground truth about knowledge characteristics [36] is also valid for software artifacts:

- Use of knowledge does not consume it.
- Transfer of knowledge does not imply losing it.
- Knowledge is abundant; the problem lies on the proper use and exploitation.
- Much of an organization's valuable knowledge walks out the door at the end of the day.

According to the current SE context, it seems that graph-approaches based on a semantic network and deployed under a set of standards in a service-oriented environment, are the most appropriate candidates. Considering this environment, the following knowledge representation paradigms (focusing on web-oriented technologies) have been selected for comparison:

1-The **Resource Description Framework** (RDF) [37] is a framework for representing information resources in the Web using a directed graph data model. The core structure of the RDF abstract syntax is a set of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. An RDF graph can be visualized as a set of nodes and directed-arcs diagram, in which each triple is represented as a node-arc-node link. RDF has been used as the underlying data model for building RDFS/OWL ontologies, gaining momentum in the web-based environment due to the explosion of the Semantic Web and Linked Data initiatives that

[2] http://stardog.com/

[3] https://www.stardog.com/blog/nasas-knowledge-graph/

aim to represent and exchange data (and knowledge) between agents and services under the web-based protocols.

2-The **RDF Schema (RDFS**) [38, p. 1] provides a data-modeling vocabulary for RDF data. It represents a first try to support the creation of formal and simple ontologies with RDF syntax. RDFS is a formal and simple ontology language in which it is possible to define class and property hierarchies, as well as domain and range constraints for properties. One of the benefits of this property-centric approach is that it allows anyone to extend the description of existing resources.

3-The **OWL (Ontology Web Language)** [39] is an ontology language for capturing meaningful generalizations about data in the Web. It includes additional constructors for building richer class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. OWL 1.1 consists of three sub-languages with different levels of expressivity: 1) OWL Lite, 2) OWL DL (Description Logics) and 3) OWL Full.

4-The **OWL 2.0** [39] family defines three different profiles: OWL 2 EL (Expressions Language), OWL 2 QL (Query Language) and OWL RL (Rule Language). These profiles represents a syntactic restriction of the *OWL 2 Structural Specification* and more restrictive than OWL DL. The use of profiles is motivated by the needs of different computational processes. OWL EL is designed for enabling reasoning tasks in polynomial time. The main aim of OWL 2 QL is to enable conjunctive queries to be answered in LogSpace using standard relational database technology. Finally, OWL 2 RL is intended to provide a polynomial time reasoning algorithm using rule-extended database technologies operating directly on RDF triples. In conclusion, OWL 2.0 adds new functionalities regarding OWL 1.x. Most of them are syntactic sugar but others offer new expressivity [39]: keys, property chains, richer datatypes, data ranges, qualified cardinality restrictions, asymmetric, reflexive, and disjoint properties; and enhanced annotation capabilities.

3-The **RIF Core** (Rule Interchange Format) [40] comprises a set of dialects to create a standard for exchanging rules among rule systems, in particular among Web rule engines. RIF was designed for exchanging rather than developing a single one-fits-all rule language. RIF dialects fall into three broad categories: first-order logic, logic-programming, and action rules. The family of dialects comprises: 1) logic-based dialects (RIF-BLD) including languages that employ some kind of logic such as First Order Logic (usually restricted to Horn Logic) or non-first-order logics; 2) rules-with-actions (RIF-PRD) dialects comprising rule systems such as Jess, Drools and JRules as well as event-condition-action rules such as Reaction RuleML. RIF also defines compatibility with OWL and serialization using RDF.

5-The **SRL** (System Representation Language) [41] [42] is based on the ground idea that whatever information can be described as a group of relationships between concepts. Therefore, the leading element of an information unit is the relationship. For example, Entity/Relationship data models are certainly represented as relationships between entity types; software object models can also be represented as relationships among objects or classes; in the process modeling area, processes can be represented as causal/sequential relationships between sub-processes. Moreover, UML (Unified Modeling Language) or SysML (System Modeling Language) metamodels can also be modeled as a set of relationships between metamodel elements.

SRL also includes a repository model to store information and relationships with the aim of reusing all kind of knowledge chunks. Furthermore, text-based information can certainly be represented as relationships between terms by means of the same structure. Indeed, to represent human language text, a set of well-constructed sentences, including the subject+verb+predicate (SVP) should be used. The SVP structure can be then considered as a relationship typed V between the S and the predicated P. In SRL, the simple representation model for describing the content of whatever artifact type (requirements, models, tests, maps, text docs or source code) is as follows:

SRL representation for artifact α = i_{α} = $\{(RSHP_1), (RSHP_2), \ldots, (RSHP_n)\}$ where every single RSHP (relationship) is called RSHP-description and must be described using terms.

One important consequence of this representation model is that there is no restriction to represent a particular type of knowledge. Furthermore, SRL has been used as the underlying information model to build general-purpose indexing and retrieval systems, domain representation models [41], approaches for quality assessment of requirements and knowledge management tools such as knowledgeMANAGER.

Obviously, a plethora of other knowledge representation mechanisms and paradigms can be found as it is presented below. However, we focus here on comparing those that satisfy the three basic ideas of this study: 1) a language for representing any artifact metadata and contents; 2) a system for indexing and retrieval and 3) a standard input/output interface (data shape+REST+RDF) to share and exchange artifact metadata and contents.

The SBVR (Semantics of Business Vocabulary and Rules). It is an OMG standard to define the basis for formal and detailed natural language declarative description of a complex entity. The Ontology Definition Metamodel (ODM). It is an OMG standard for knowledge representation, conceptual modeling, formal taxonomy development and ontology definition. It enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF.

ODM-based ontologies can be used to support: 1) interchange of knowledge; 2) representation of knowledge in ontologies and knowledge bases; and 3) specification of expressions that are the input to, or output from, inference engines. The Reusable Asset Specification (RAS), an OMG standard that addresses the engineering elements of reuse. It attempts to reduce the friction associated with reuse transactions through consistent, standard packaging.

V. Evaluation and Discussion

The previous section has reviewed the main approaches for knowledge representation in a web-oriented environment for exchanging software artifacts. As a result, Table 2 (in the Annex) shows the main characteristics and capabilities that can be found in RDF, RDFS, OWL and SRL with special focus on those regarding knowledge management and, more specifically knowledge representation. In order to select the proper mechanism for knowledge representation of software artifacts, the following points must be considered:

- RDF is based on a directed graph and it can only represent binary relationships (unless reification and blank nodes are used). As a representation mechanism, RDF presents some restrictions that have been outlined in several works [24] . For instance, N-ary relationships [25], practical issues dealing with reification [26] and blank nodes [27] are well-known RDF characteristics that do not match the needs of a complete framework for knowledge representation. Furthermore, RDF is built on two main concepts: resources and literals. However, a literal value cannot be used as the subject of an RDF triple. Although this issue can be overcome using a blank node (or even reification) and the property `rdf:value`, it adds extra complexity for RDF users. Finally, RDF has been designed to represent logical statements, constraining also the possibility of representing other widely used paradigms such as objects or entity-relationships models. Due to these facts, it seems clear that RDF can be used for exchanging data, but it is not the best candidate for knowledge representation.

- RDFS is a good candidate for modeling lightweight formal ontologies including some interesting capabilities close to object-oriented models. RDFS ontologies can be serialized as RDF, but this feature can also be a disadvantage due to expressivity restrictions of RDF. Again, RDFS has been designed for expressing logical statements that describe web resources, so its use for other types of information seems to be not advisable.

- Building on the previous discussion, OWL presents a family of logic dialects for knowledge representation. It is based on strong logic formalisms such as Description Logic or F-Logic. It was also designed for asserting facts about web resources although it can be used as a general logic framework for any type of knowledge. One of the main advantages of OWL is the possibility of performing reasoning processes to check consistency or infer types. However, reasoning can be considered harmful in terms of performance and most of times it is not necessary when data is being exchanged. Besides, OWL is not the best candidate for data validation, a key process in knowledge exchange.

- RIF Core and the family of RIF dialects have been included in this comparison due to the fact that most of domain knowledge is embedded in rules. However, RIF was not designed for data validation and its acceptance is still low (just a few tools export RIF and less are capable of importing RIF files). On the other hand, RIF makes use of the web infrastructure to exchange rules, what means also that this environment is a very good candidate to exchange data, information and knowledge.

- SRL, based on relationships, allows domain experts to create relationships between terms, concepts or even artifacts (containers). It provides a framework for knowledge representation with capabilities for expressing any kind of cardinality and N-ary relationships. SRL is based on undirected property graphs, enhancing expressivity. Although it has not been directly designed for data validation, its metamodel allows the possibility of checking cardinality, value constraints, domain and range restrictions. One of the strong points of SRL is the native support of a tool such as knowledgeMANAGER and the possibility of automatically providing semantic indexing and retrieval mechanisms. Both have generated a relevant acceptance in the industry for authoring and quality checking.

Finally, as a general comment, there is also a lack of tools working natively on RDF. Furthermore, RDF was conceived to exchange information over the web. Although some RDF repositories can provide capabilities for indexing and searching RDF resources through an SPARQL interface, the experience has demonstrated that most of times RDF is translated into the native data model of a tool.

Based on this evaluation and considering the three basic requirements of this review, we conclude that RDF is a good alternative to exchange data. Since formal ontologies and reasoning processes are not completely necessary and, instead, data validation is a key aspect for boosting interoperability and reuse of software artifacts, it also seems clear that an approach like SRL can perfectly fit to the major objective of knowledge representation in the SE discipline. However, we recognize that the use of formal RDFS or OWL ontologies is not incompatible with data shapes, but possible. RDFS and OWL are languages for building domain vocabularies, while SRL is already a domain vocabulary for knowledge representation, so it is possible to define SRL in a formal RDFS/OWL ontology.

VI. Conclusions and Future Work

The application of the Linked Data principles to exchange data in the software development lifecycle is gaining momentum. Software artifacts reuse via interoperability is a key enabler for boosting collaboration in the development of complex software systems. The concept of continuous engineering is becoming a reality since it is possible to integrate data and services under common protocols and data models.

In this context, the capability of reusing existing software artifacts is a key factor that can ease teams to develop systems faster and safer. However, software is not anymore, a piece of logical instructions but a kind of knowledge and organizational asset. That is why a proper environment for knowledge management of software artifacts should provide the appropriate mechanisms for representing, storing, indexing and retrieving any kind of software artifact. However, some approaches such as OSLC relying on Linked Data principles cannot easily deployed due to issues regarding expressivity in RDF or the need of tools for easily process RDF (mainly from a developer perspective). In this context, the use of well-documented REST APIs (e.g. based on standards such the OpenAPI Specification) are good enough to access and exchange data generated during the development lifecycle.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Celtic Next-EUREKA initiative under code Nº C2017/3-2-IoD (Internet of DevOps) and from specific national programs and/or funding authorities.

REFERENCES

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.

[2] M. Kersten, "Five Predictions for the Coming Decades of Software," *IEEE Softw.*, vol. 35, no. 5, pp. 7–9, Sep. 2018.

[3] P. Bourque, R. E. Fairley, and others, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[4] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee Softw.*, vol. 29, no. 6, pp. 18–21, 2012.

[5] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1498–1516, 2013.

[6] I. Nonaka and H. Takeuchi, *The knowledge-creating company: How japanese companies create the dynamics of innovation*. New York: Oxford University Press, 1995.

[7] R. Hull and R. King, "Semantic database modeling: Survey, applications, and research issues," *ACM Comput. Surv. CSUR*, vol. 19, no. 3, pp. 201–260, 1987.

[8] R. Davis, H. Shrobe, and P. Szolovits, "What is a knowledge representation?," *AI Mag.*, vol. 14, no. 1, p. 17, 1993.

[9] T. Groza, S. Handschuh, T. Clark, S. Buckingham Shum, and A. de Waard, "A short survey of discourse representation models," 2009.

[10] K. Manikas and K. M. Hansen, "Software ecosystems – A systematic literature review," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1294–1306, May 2013.

[11] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

[12] I. Sommerville, *Software engineering*, Tenth edition. Boston: Pearson, 2016.

[13] C. Ebert, "50 Years of Software Engineering: Progress and Perils," *IEEE Softw.*, vol. 35, no. 5, pp. 94–101, Sep. 2018.

[14] R. C. Martin, Ed., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

[15] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[16] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synth. Lect. Softw. Eng.*, vol. 1, no. 1, pp. 1–182, Sep. 2012.

[17] P. Clements and L. Northrop, *Software product lines: practices and patterns*. Boston: Addison-Wesley, 2002.

[18] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[19] L. Bass, I. M. Weber, and L. Zhu, *DevOps: a software architect's perspective*. New York: Addison-Wesley Professional, 2015.

[20] V. R. Benjamins, D. Fensel, and A. Gómez-Pérez, "Knowledge Management through Ontologies," in *PAKM*, 1998.

[21] K. Baclawski, M. M. Kokar, R. J. Waldinger, and P. A. Kogut, "Consistency Checking of Semantic Web Ontologies," in *International Semantic Web Conference*, 2002, pp. 454–459.

[22] C. Bizer and R. Cyganiak, "Quality-driven information filtering using the WIQA policy framework," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 7, no. 1, pp. 1–10, Jan. 2009.

[23] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker, "An empirical survey of Linked Data conformance," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 14, pp. 14–44, Jul. 2012.

[24] S. Powers, *Practical RDF*. Beijing ; Sebastopol: O'Reilly, 2003.

[25] N. Noy and A. Rector, "Defining N-ary Relations on the Semantic Web," W3C Working Group, 2006.

[26] V. Nguyen, O. Bodenreider, and A. Sheth, "Don't like RDF reification?: making statements about statements using singleton property," 2014, pp. 759–770.

[27] A. Mallea, M. Arenas, A. Hogan, and A. Polleres, "On blank nodes," in *The Semantic Web–ISWC 2011*, Springer, 2011, pp. 421–437.

[28] A. G. Ryman, A. L. Hors, and S. Speicher, "OSLC Resource Shape: A language for defining constraints on Linked Data," in *LDOW*, 2013.

[29] K. Coyle and T. Baker, "Dublin Core Application Profiles Separating Validation from Semantics," W3C, Mar. 2013.

[30] D. Kontokostas *et al.*, "Test-driven evaluation of linked data quality," in *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, 2014, pp. 747–758.

[31] I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'hommeau, H. R. Solbrig, and S. Staworko, "Validating RDF with Shape Expressions," *CoRR*, vol. abs/1404.1270, 2014.

[32] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, "Validating RDF Data," *Synth. Lect. Semantic Web Theory Technol.*, vol. 7, no. 1, pp. 1–328, Sep. 2017.

[33] J. Alvarez-Rodríguez, J. Labra-Gayo, and P. Ordoñez de Pablos, "Leveraging Semantics to Represent and Compute Quantitative Indexes: The RDFIndex Approach," in *Metadata and Semantics Research*, vol. 390, E. Garoufallou and J. Greenberg, Eds. Springer International Publishing, 2013, pp. 175–187.

[34] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *J Web Sem*, vol. 5, no. 2, pp. 51–53, 2007.

[35] Holger Knublauch, James A. Hendler, and Kingsley Idehen, "SPIN - Overview and Motivation," W3C, Member Submission, Feb. 2011.

[36] D. Morey, M. T. Maybury, and B. M. Thuraisingham, *Knowledge management: classic and contemporary works*. Cambridge, Mass.: MIT Press, 2002.

[37] P. Hayes, "RDF Semantics," World Wide Web Consortium, Feb. 2004.

[38] D. Brickley and R. V. Guha, Eds., *RDF Schema 1.1*. W3C Recommendation, 2014.

[39] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer," World Wide Web Consortium, W3C Recommendation, Oct. 2009.

[40] H. Boley, G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds, Eds., *RIF Core Dialect (Second Edition)*. W3C Recommendation, 2013.

[41] I. Díaz, J. Llorens, G. Genova, and J. M. Fuentes, "Generating domain representations using a relationship model," *Inf. Syst.*, vol. 30, no. 1, pp. 1–19, Mar. 2005.

[42] J. M. Alvarez-Rodríguez, J. Llorens, M. Alejandres, and J. M. Fuentes, "OSLC-KM: A knowledge management specification for OSLC-based resources," *INCOSE Int. Symp.*, vol. 25, no. 1, pp. 16–34, Oct. 2015.

Annex I

Feature	RDF [37]	RDFS [38, p. 1]	OWL [39]	RIF Core [40]	SRL [41]
Full Name	Resource Description Framework	Resource Description Framework Scheme	Ontology Web Language	Rule Interchange Format	System Representation Language
First Version	1.0 (February 2004)	1.0 (February 2004)	1.0 (February 2004)	First edition (December 2012)	v1 (January 2004)
Last version	1.1 (February 2014)	1.1 (February 2014)	2.0 (December 2012)	Second edition (February 2013)	v14 (January 2015)
Designed for	Representation of logical statements	Data modeling vocabulary for RDF data	Formal ontology design	Definition of Horn rules	Representation of relationships between knowledge items
Target use	Data exchange of facts, rules and ontologies	Data model	Ontology creation	Rule interchange	Universal knowledge representation and re-use
Data model	Directed graph	Directed graph	Directed graph	Object Model	Undirected (property) graph
Underlying semantics	RDF formal semantics	RDFS Semantics	OWL 2. Direct Semantics and RDF-based Semantics	RIF Core Semantics	Explicit metamodel
Expressivity	Simple RDF triples (s, p, o) to represent binary relationthips.	Classes (sub and super classes) and Properties (domain and ranges)	OWL 1.1: • DL (Description Logic), • Lite, • Full OWL 2.0: • EL (Expressions Language) • QL (Query Language) • RL (Rule Language)	• RIF-Core (Core Dialect) • RIF-BLD (Basic Logic Dialect) • RIF-PRD (Production Rule Dialect) • RIF-FLD (Framework for Logic Dialects) • RIF-OWL 2 RL and RIF RDF • RIF XML	Any kind of relationship (**SVP**). • N-ary relationships. • Non logic formalism. • Knowledge containers. (reification)
Validation	RDF Data Shapes: • OSLC Resource Shapes, SHACL (Shapes Constraint Language) • SheX (Shape Expressions) • SPIN (SPARQL Inferencing Notation) and SPARQL Rules	Semantic reasoning + see RDF	Semantic reasoning + see RDF	Metamodel conformity	Metamodel conformity
Inference	Not at graph level.	Yes, but restricted to type inference and super/sub classes and properties	Yes, depending on the underlying logic formalism: First Order Logic, F-Logic, DL, etc.	Yes	Not at graph level.
Identifiers	URIs (HTTP URIs if Linked Data). Unique Name Assumption (UNA).	See RDF	See RDF	Internal IDs and UNA.	Internal IDs and UNA.
Access protocol	HTTP-based (REST resources)	See RDF	See RDF	See RDF and native APIs	Native API
Query language	SPARQL and RDQL	See RDF	SWRL	XPATH (if XML is used as serialization format)	RSHP query language
Storage	RDF repository (native RDF repositories, graph-based databases, and wrappers on top of existing relational databases)	See RDF	See RDF	Native API	SQL or NonSQL database
Formats (syntax)	RDF/XML, JSON, Turtle, N3, Manchester	See RDF	See RDF	XML	RDF/XML, ISO 25964-"The international standard for thesauri and interoperability with other vocabularies", etc.
Visualization	RDF visualization libraries such as Allegro graph or RDFgravity and other general-purpose graph visualization frameworks Graphviz, Touchgraph, Gephi, Cytoscape, D3.js.	See RDF	See RDF	Native Rule IDEs	RSHP visualization language and the aforementioned general-purpose graph visualization frameworks.
Application	Integration of databases, applications and services through a common and shared data model.	See RDF	See RDF	Interchange of business rules and connection with existing ontologies	Semantics-based information retrieval using a natural language interface to support other services such as traceability or quality.
Status	W3C recommendation	W3C recommendation	W3C recommendation	W3C recommendation	Industry-oriented
Tools	Protégé, SWOOP or Terminae, TopBraid Composer (ontology editors)	See RDF and RDFS reasoners such as Pellet, Racer or Jess	See RDFS	JRules, Drools or Jess (mainly exporters not importers)	knowledgeMANAGER (a complete suite for knowledge management with RDF import/export capabilities)

Table 2 A comparison among the main approaches for knowledge representation using an underlying semantic network.