

Studies on the Semantic Web

Ontology Engineering with Ontology Design Patterns

Foundations and Applications

Pascal Hitzler, Aldo Gangemi,
Krzysztof Janowicz, Adila Krisnadhi,
Valentina Presutti (Eds.)

ONTOLOGY ENGINEERING WITH ONTOLOGY DESIGN PATTERNS

Semantic Web has grown into a mature field of research. Its methods find innovative applications on and off the World Wide Web. Its underlying technologies have significant impact on adjacent fields of research and on industrial applications. This new book series reports on the state of the art in foundations, methods, and applications of Semantic Web and its underlying technologies. It is a central forum for the communication of recent developments and comprises research monographs, textbooks and edited volumes on all topics related to the Semantic Web.

Editor-in-Chief:
Pascal Hitzler

Editorial Board:

Diego Calvanese, Vinary Chaudhri, Fabio Ciravegna, Michel Dumontier, Dieter Fensel,
Fausto Giunchiglia, Carole Goble, Asunción Gómez Pérez, Frank van Harmelen,
Manfred Hauswirth, Ian Horrocks, Krzysztof Janowicz, Michael Kifer, Riechiro Mizoguchi,
Mark Musen, Daniel Schwabe, Barry Smith, Steffen Staab, Rudi Studer

Publications:

Vol. 025 – Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, Valentina Presutti (Eds.), Qntology Engineering with Ontology Design Patterns: Foundations and Applications

Vol. 024 Olaf Hartig, Querying a Web of Linked Data. Foundations and Query Execution

Vol. 023 Natalia A. Diaz Rodriguez, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living

Vol. 022 Juan F. Sequeda, Integrating Relational Databases with the Semantic Web

Vol. 021 Laurens Rietveld, Publishing and Consuming Linked Data

Vol. 020 Tom Narock and Peter Fox (Eds.), The Semantic Web in Earth and Space Science. Current Status and Future Directions

Vol. 019 Aidan Hogan, Reasoning Techniques for the Web of Data

Vol. 018 Jens Lehmann and Johanna Völker (Eds.), Perspectives on Ontology Learning

Vol. 017 Jeff Z. Pan and Yuting Zhao (Eds.): Semantic Web Enabled Software Engineering

Vol. 016 Gianluca Demartini, From People to Entities: New Semantic Search Paradigms for the Web

Vol. 015 Carlos Buil-Aranda, Federated Query Processing for the Semantic Web

Vol. 014 Sebastian Rohjans, Semantic Service Integration for Smart Grids

Vol. 013 Tudor Groza, Advances in Semantic Authoring and Publishing

Vol. 012 Boris Villazón-Terrazas, A Method for Reusing and Re-engineering Non-ontological Resources for Building Ontologies

Vol. 011 Christoph Lange, Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration

Vol. 010 Tran Duc Than, Process-oriented Semantic Web Search

Ontology Engineering with Ontology Design Patterns

Foundations and Applications

Edited by

Pascal Hitzler

Aldo Gangemi

Krzysztof Janowicz

Adila Krisnadhi

Valentina Presutti

Pascal Hitzler, Wright State University, U. S. A.

Aldo Gangemi, ISTC-CNR Roma, Italy

Krzysztof Janowicz, University of California, Santa Barbara, U. S. A.

Adila Krisnadhi, Wright State University, U. S. A. and Universitas Indonesia

Valentina Presutti, ISTC-CNR Roma, Italy

IOS
Press



Pascal Hitzler, pascal.hitzler@wright.edu
Aldo Gangemi, aldo.gangemi@gmail.com
Krzysztof Janowicz, janowicz@ucsb.de
Adila Krisnadhi, krisnadhi@gmail.com
Valentina Presutti, valentina.presutti@cnr.it

Bibliographic Information published by the Deutsche Nationalbibliothek:
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie.
Detailed bibliographic data are available on the Internet at <https://portal.d-nb.de>.

<i>Publisher</i>	<i>Co-Publisher</i>
Akademische Verlagsgesellschaft AKA GmbH P.O. Box 22 01 16 14061 Berlin Germany Tel.: 0049 (0)30 36 43 01 64 info@aka-verlag.de www.aka-verlag.com	IOS Press BV Nieuwe Hemweg 68 1013 BG Amsterdam The Netherlands www.iospress.nl

© 2016, Akademische Verlagsgesellschaft AKA GmbH, Berlin

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior permission from the publisher.

Reproduced from PDF supplied by the author
Printer: Bookstation GmbH, Anzing
Printed in Germany

ISSN 1868-1158 (print) / ISSN 2215-0870 (online)
ISBN 978-3-89838-715-6 (AKA)
ISBN 978-1-61499-675-0 (IOS Press, print) / 978-1-61499-676-7 (IOS Press, online)

Preface

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Aldo Gangemi, LIPN, Université Paris 13, CNRS UMR7030 and ISTC-CNR, Italy

Krzysztof Janowicz, University of California, Santa Barbara, USA

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

Valentina Presutti, STLab, ISTC-CNR, Italy

Patterns in general can be defined as invariances across observed data, objects, processes, etc. Patterns in the Semantic Web may emerge from data, ontologies, as well as from procedural aspects of design at either the modelling or implementation level.

Design patterns have emerged in computer science from the pioneering architectural work of Christopher Alexander [1], firstly applied to software engineering [3], then to workflows [11], HCI [10], data modelling [8], knowledge engineering [2], and eventually the Semantic Web [4, 5, 9], where they are known as *Ontology Design Patterns* (ODP), *knowledge patterns*, or *linked data patterns*, according to the community that uses them (e.g. ontology designers, knowledge engineers, linked data publishers, etc.). The main innovation of design patterns is their *critical* approach to compare possible solutions against recurrent problems: alternatives, pros and cons, openness to change, examples and counterexamples, lessons learnt, etc.

In this book, we present a broad spectrum of theories, experiences, and models that focus on ontology design patterns. We have collected them into four parts: *Foundations*, *Practice*, *Selected Examples*, and an appendix providing a primer on RDF and OWL.

The Foundations part includes chapters that provide a comprehensive view of methods, modelling examples, generic patterns, as well as relevant research questions that are currently open. This part opens with a modelling example using patterns in the domain of chess, followed by a summary of the eXtreme Design methodology that combines an agile approach with pair programming-inspired methods, scenario analysis, competency questions, [7] and matching them to patterns.

The next chapter faces the problem of defining and assessing the quality of design patterns, followed by a chapter on the role of logical axiomatization in ontology design patterns, and some related research issues.

Next, three chapters on generic patterns extracted from the foundational ontology DOLCE, from the Descriptions and Situations framework, and about pattern languages as known in conceptual modelling are then followed by a presentation of anti-patterns, and finally by a collection of research questions provided by the book authors and beyond. That chapter is important as a spin-off to further research in the field, covering e.g.: features, qualities, languages and standards for ontology design patterns, methods for using, constructing, and extracting ODPs, tooling and infrastructures (support for reuse, repositories, versioning, sustainability), and modularization based on ODPs.

In the Foundations part, at least two chapters introduce *Content Ontology Patterns*, also known as *Knowledge Patterns*, [2, 5] which are reusable components that can be used to match competency questions. We remark that while our initial definition of patterns as invariances across observed data, objects, processes applies to any kind of pattern, we need to distinguish the purely symbolic patterns of mathematical pattern science [6], as studied in data mining, machine learning, complex systems, etc., from the knowledge patterns. Knowledge patterns are not only symbolic: they also have a semantic interpretation, be it formal, or cognitive. Such interpretation consists in the meaning of a pattern, e.g. a type of fact reported in news, a kind of soccer event from a picture, an aggressive attitude in a sentence, etc.

However, knowledge patterns are not confined to foundational, i.e. domain-independent theories, but they can be extracted or may emerge in particular domains or for particular tasks. The two following parts of the book describe nine such examples. On one hand, the Practice part contains mainly procedural patterns: Linked Data publishing patterns are presented, followed by examples and lessons learnt when working with domain experts, when using patterns for ontology transformation, for data integration, and for tracing supply chains. On the other hand, the Selected Examples section drills down into applications of patterns for domains dealing with information entities, roles, spatial trajectories, and particle detector final states in physics.

The editors believe that the book spans across most of the relevant examples, areas, and research issues concerning ontology design patterns, of course without aiming at completeness, which is intrinsically denied by pattern-based design: we will always get new problems and solutions, and a space of possible design choices is sparsely populated (the reason why we can perceive or construct patterns), because of the inherent economy of human cognition, which tries to find shortcuts that can be repeatedly applied in order to optimise competences, export them by analogy, and finally exploit them for the individual or common good.

Bibliography

- [1] C. Alexander. *The Timeless Way of Building*. Oxford Press, 1979.
- [2] P. Clark, J. Thompson, and B. Porter. Knowledge Patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge*

- Representation and Reasoning*, pages 591–600, San Francisco, 2000. Morgan Kaufmann.
- [3] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
 - [4] A. Gangemi. Ontology design patterns for semantic web content. In Y. G. et al., editor, *Proc. of the 4th Int. Semantic Web Conference, ISWC 2005*, volume 3729 of *LNCS*, pages 262–276. Springer, 2005.
 - [5] A. Gangemi and V. Presutti. Towards a pattern science for the semantic web. *Semantic Web*, 1(1-2):61–68, 2010.
 - [6] U. Grenander. *Elements of Pattern Theory*. Johns Hopkins University Press, Baltimore, 1996.
 - [7] M. Gruninger and M. S. Fox. The Role of Competency Questions in Enterprise Engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.
 - [8] D. C. Hay. *Data Model Patterns*. Dorset House Publishing, 1996.
 - [9] V. Presutti and A. Gangemi. Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In *ER '08: Proceedings of the 27th International Conference on Conceptual Modeling*, pages 128–141, Berlin, Heidelberg, 2008. Springer-Verlag.
 - [10] J. Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly, April 2007.
 - [11] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.

This page intentionally left blank

Contents

Introduction: Ontology Design Patterns in a Nutshell	xi
Krzysztof Janowicz, Aldo Gangemi, Pascal Hitzler, Adila Krisnadhi, Valentina Presutti	
Part I. Foundations of Ontology Design Patterns	
1. Modeling With Ontology Design Patterns: Chess Games As a Worked Example	3
Adila Krisnadhi, Pascal Hitzler	
2. Engineering Ontologies with Patterns – The eXtreme Design Methodology	23
Eva Blomqvist, Karl Hammar, and Valentina Presutti	
3. Quality of Content Ontology Design Patterns	51
Karl Hammar	
4. On the Roles of Logical Axiomatizations for Ontologies	73
Pascal Hitzler, Adila Krisnadhi	
5. Dolce+D&S Ultralite and its main ontology design patterns	81
Valentina Presutti, Aldo Gangemi	
6. Multi-layered n-ary Patterns	105
Aldo Gangemi, Valentina Presutti	
7. Ontology Pattern Languages	133
Ricardo Falbo, Monalessa Barcellos, Fabiano Ruy, Giancarlo Guizzardi, Renata Guizzardi	
8. Anti-patterns in Ontology-driven Conceptual Modeling: The Case of Role Modeling in OntoUML	161
Tiago Sales, Giancarlo Guizzardi	
9. Collected Research Questions Concerning Ontology Design Patterns	189
Karl Hammar, Eva Blomqvist, David Carral, Marieke van Erp, Antske Fokkens, Aldo Gangemi, Willem Robert van Hage, Pascal Hitzler, Krzysztof Janowicz, Nazifa Karima, Adila Krisnadhi, Tom Narock, Roxane Segers, Monika Solanki, Vojtech Svatek	
Part II. Ontology Design Patterns In Practice	
10. Ontology Design Patterns for Linked Data Publishing	201
Adila Krisnadhi, Nazifa Karima, Pascal Hitzler, Reihaneh Amini, Víctor Rodríguez-Doncel, Krzysztof Janowicz	
11. Modeling Ontology Design Patterns with Domain Experts – A View From the Trenches	233
Krzysztof Janowicz	

12. Using ODPs for Ontology Transformation	245
Vojtěch Svátek, Ondřej Zamazal, Marek Dudáš	
13. Ontology Design Patterns for Data Integration: The GeoLink Experience	267
Adila Krisnadhi	
14. Towards Enabling Traceability in Supply Chains via Ontology Design Patterns	279
Monika Solanki	

Part III. Selected Examples of Ontology Design Patterns

15. The Information Realization Pattern	299
Aldo Gangemi, Silvio Peroni	
16. The Role Patterns	313
Adila Krisnadhi	
17. The Semantic Trajectory Pattern	321
Yingjie Hu, Krzysztof Janowicz	
18. An Ontology Design Pattern for Detector Final States	329
David Carral	

Appendix

A. A Primer on RDF and OWL	337
Frederick Maier	
Index	363

Introduction: Ontology Design Patterns in a Nutshell

Krzysztof Janowicz, University of California, Santa Barbara, USA

Aldo Gangemi, LIPN, Université Paris 13, CNRS UMR7030 and ISTC-CNR, Italy

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

Valentina Presutti, STLab, ISTC-CNR, Italy

Patterns in general can be defined as distinctive and repetitive invariants across observed data, objects, processes, and so forth, that are either manmade or occur naturally. Design patterns have emerged in computer science from the pioneering architectural work of Christopher Alexander [1], firstly applied to software engineering [9], then to workflows [31], HCI [30], data modeling [17], knowledge engineering [6], and eventually the Semantic Web [10, 13], where they are known as *Ontology Design Patterns* (ODP), *knowledge patterns*, or *linked data patterns*, depending on the community that uses them, e.g., ontology designers, knowledge engineers, linked data publishers, and so forth.

Patterns on the Semantic Web typically emerge from (linked) data, ontologies, and queries, as well as from procedural aspects of design at either the modeling or implementation stage. The main innovation of design patterns lies in the observation that a majority of datasets and ontologies share a relatively small set of common modeling and publishing challenges that can be approached by a common strategy, established best practice, or by combining existing building blocks. In principle, there is a distinction between design patterns as critical reviews of alternative ways of modeling something, such as a review of alternatives to represent an n -ary relation with $n \geq 2$ in RDF, vs. design patterns as reusable components to create a product, such as the description of a reusable ontology to represent such an n -ary relation in OWL. However, a proper design pattern should always mention its motivating requirements, applicability limits, benefits and shortcomings, and so forth. Therefore, whether it focuses on alternative methods, or on one or multiple alternative reusable components, a design pattern always provides a critical approach to deal with recurrent problems. Finally, it

is worth pointing out that the initial definition of patterns as invariants applies to any kind of pattern, while it makes sense to distinguish the purely symbolic patterns of mathematical pattern science [14], as studied in data mining, machine learning or complex systems, from knowledge patterns. Knowledge patterns are not merely symbolic, they also have a semantic interpretation, be it formal, or cognitive; e.g., a type of fact reported in the news.

While ontology design patterns can be considered analogous to software design patterns known from object oriented modeling, there are also clear differences. For instance, while ODPs and software design patterns can act as strategies, there is no clear design pattern counterpart for so-called *content ODPs* (knowledge patterns). These are typically modeled for frequently reoccurring aspects of more complex ontologies and thus act as building blocks rather than strategies. Examples include the *trajectory pattern* [18] that can be used in ontologies that model the transportation domain, wildlife monitoring, scientific cruises, to name but a few. Such building blocks, however, are not confined to domain-specific cases. For example, the *information realization pattern* [25] addresses the common problem of describing the relation between an information object, e.g., a book, and its physical realization, i.e., the specific paper copy of said book.

While the number of strategy patterns, such as the *structural n-ary relation pattern*, is relatively small, there is a wide and growing variety of knowledge patterns, and no immediate reason to establish an upper bound for their number. This leads to an interesting question, namely whether and how knowledge design patterns can be distinguished from other *small* ontologies. There are many possible criteria (cf. [10, 13, 25] for discussions). Ideally, ontology design patterns should be extendable but self-contained, minimize ontological commitments to foster reuse, address one or more explicit requirements (or use cases, competency questions), be associatable to an ontology unit test [33], be the representation of a core notion in a domain of expertise (so-called “blinking effect”, [13]), be grounded in conceptual or lexical frames [24], be alignable to other patterns, span more than one application area or domain, address a single invariant instead of targeting multiple reoccurring issues at the same time, follow established modelling best practices, and so forth.

While these criteria imply that there is a smooth transition between small ontologies and patterns, they also highlight the fact that ontology design pattern research itself is related to numerous other areas of study such as ontology modularization [8], ontology alignment [28], Linked Data publishing [27], knowledge extraction [24], knowledge discovery [22], ontology design [3], foundational ontologies, and so on. In many of the aforementioned cases, OPDs have provided new perspectives and insights to fuel ongoing research. In addition, patterns have also served as minimal core components of widely used ontologies such as the Semantic Sensor Network ontology [7]. Finally, some influential ontologies, such as the Simple Event Model [32] have either inspired modern ODP research, our could be perceived as patterns themselves.

This leads to another interesting question, namely why we are recently witnessing a rapid increase in the publication and reuse of patterns more than 10 years after their initial excogitation. We believe that this is due to a combination of several factors, more specifically changes in the Semantic Web research land-

scape. Early work on the Semantic Web (ca. 2001-2008) was heavily driven by foundational investigations of information ontologies and knowledge representation languages. Consequently, the focus was rather on describing which ontological choices can and should be made in partitioning the world, leading to questions of how to distinguish objects from processes, designing well modularized core ontologies in, say, the medical or legal domains [11, 12], or identifying *anti-patterns* for educational and model-checking purposes [5, 23, 26]. In many respects, the early Linked Data work (ca. 2008-2012) can be seen as a direct counter-reaction to address those problems by proposing minimal vocabularies and heavily restricting the expressivity of used knowledge representation languages, to focus on scalability. While this led to a rapidly growing ecosystem of interconnected data hubs, the early Linked Data Web could not fulfil some of the key promises it was set out to address – federated queries across distributed query endpoints being the most prominent of them. In a nutshell, *linked data* without semantics turned out to be merely *more data* [2, 13, 19], and required a detailed manual inspection and familiarity with the used vocabularies, lineage, and so forth, to be used meaningfully. Similar observations were also made with respect to knowledge extraction [24] and the creation of knowledge graphs and query answering [20] – *raw* data alone is not sufficient. Intuitively, one may assume that this led to a revival of the early work on foundational (top-level) ontologies. By that time, however, the Semantic Web landscape had changed due to the availability of hundreds of highly heterogeneous Linked Data sourced from a multitude of domains [29]. In such a setting, where *synthesis* becomes a crucial task, semantic interoperability has to be fostered without restricting heterogeneity [21]. This, however, is one of the key strengths of ontology design patterns.

By offering common strategies and building blocks, ontology design patterns act as an interoperability fallback level [4] through which local conceptualizations can differ to a degree required to appropriately model a given domain or application while still sharing a common conceptual core. To give a concrete example, imagine three data providers and their respective ontologies. The first provider offers data about (pedestrian) human mobility captured using smartphones, other mobile devices, and social media. The second provider has a broader view on transportation and offers data about cars, buses, taxis, trucks, and so forth. Finally, the third provider stores sparse GPS-based wildlife tracking data from Californian mountain lions. Further assume that the three used ontologies include the semantic trajectory pattern [18] as a common core component. The pattern models trajectories by fixes and segments between them, i.e., as a scalable discretization of the movement of some entity through space and time. Other than that, the ontologies may vary greatly. For example, the wildlife ontology does not model the *mode of transportation* and all location fixes are derived from GPS collars. In contrast, the pedestrian ontology uses so called *geo-social check-ins* to determine a human’s location, i.e., all fixes are related to some meaningful place, e.g., a restaurant. Finally, the transportation ontology may introduce a type hierarchy for the segments by adding classes such as roads and highways. Clearly, there is some data that cannot be retrieved from all providers via a federated query, e.g., only the pedestrian dataset can be queried for place types. Nonetheless, there may be many cases in which federated queries over these data can be

desirable, e.g., to detect spots where wildlife crosses highways or enters human settlements. Such queries remain indeed possible despite major changes in the used ontologies, as they only require the core classes (e.g., fixes, segments, positioning technologies) defined by the trajectory pattern. Summing up, by contributing to semantic interoperability research, knowledge extraction, query answering, and so forth, patterns fill an important gap towards a more sustainable and easier to use Linked Data Web.

So far, we have briefly introduced ontology design patterns, positioned them within the field of Semantic Web research, reviewed potential reasons for their increasing relevance, and discussed a concrete example for their usage in heterogeneous data infrastructures. Next, we will highlight current and future research issues, which will be taken up again in Chapter 9.

Starting with barriers that currently prevent a wider adoption of patterns, one key issue is the lack of an easy to use and up-to-date repository of quality controlled patterns together with a documentation and bundles of patterns that are tested to work seamlessly together [4]. Some of these issues can be addressed by a new version of <http://www.ontologydesignpatterns.org>, while others will require more research such as the combination of patterns into tested bundles/distributions. This also relates to the broader question of how to determine, and ideally measure, the quality of ontologies and ODPs in specific [15, 34]. Finally, many popular ontologies have been developed before patterns played a substantial role, and could be restructured in order to split them into earlier reusable and maintainable patterns. We believe that overcoming these boundaries would lead to a surge in the adoption of patterns for newly published as well as established Linked Datasets.

Looking forward, there is a number of next research steps that deserve further attention. Examples include the development of an ontology design pattern representation language, a set of formal relations between patterns, semantic *shortcuts* and *views* to provide different application-driven perspectives on pattern collections, more robust ways to handle issues of granularity in ODP modeling and usage, large-scale applications and deployments of patterns to modern data infrastructures, further work on pattern-based ontology alignment, knowledge pattern extraction and discovery, and finally the integration of ODPs in major tools such as ontology editors [16, 23].

Bibliography

- [1] C. Alexander. *The Timeless Way of Building*. Oxford Press, 1979.
- [2] S. Bechhofer, I. Buchan, D. De Roure, P. Missier, J. Ainsworth, J. Bhagat, P. Couch, D. Cruickshank, M. Delderfield, I. Dunlop, et al. Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2):599–611, 2013.
- [3] E. Blomqvist, A. Gangemi, and V. Presutti. Experiments on pattern-based ontology design. In *Proceedings of the fifth international conference on Knowledge capture*, pages 41–48. ACM, 2009.
- [4] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and

- M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2015.
- [5] V. K. Chaudhri, R. Katragadda, J. Shrager, and M. Wessel. Inconsistency monitoring in a large scientific knowledge base. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 66–79. Springer, 2014.
 - [6] P. Clark, J. Thompson, and B. Porter. Knowledge Patterns. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 591–600, San Francisco, 2000. Morgan Kaufmann.
 - [7] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25 – 32, 2012.
 - [8] M. d’Aquin, A. Schlicht, H. Stuckenschmidt, and M. Sabou. Criteria and evaluation for ontology modularization techniques. In *Modular ontologies*, pages 67–89. Springer, 2009.
 - [9] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
 - [10] A. Gangemi. Ontology design patterns for semantic web content. In Y. Gil et al., editors, *Proc. of the 4th Int. Semantic Web Conference, ISWC 2005*, volume 3729 of *LNCS*, pages 262–276. Springer, 2005.
 - [11] A. Gangemi. Design patterns for legal ontology constructions. *LOAIT*, 2007:65–85, 2007.
 - [12] A. Gangemi, C. Catenacci, and M. Battaglia. Inflammation ontology design pattern: an exercise in building a core biomedical ontology with descriptions and situations. *Studies in health technology and informatics*, pages 64–80, 2004.
 - [13] A. Gangemi and V. Presutti. Towards a pattern science for the semantic web. *Semantic Web*, 1(1-2):61–68, 2010.
 - [14] U. Grenander. *Elements of Pattern Theory*. Johns Hopkins University Press, Baltimore, 1996.
 - [15] K. Hammar. *Towards an Ontology Design Pattern Quality Model*. PhD thesis, Linköping University Electronic Press, 2013.
 - [16] K. Hammar. Ontology design patterns in WebProtégé. In *14th International Semantic Web Conference (ISWC-2015)*. CEUR Workshop Proceedings, 2015.
 - [17] D. C. Hay. *Data Model Patterns*. Dorset House Publishing, 1996.
 - [18] Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitzler, M. Dean, and D. Kolas. A geo-ontology design pattern for semantic trajectories. In *Spatial Information Theory*, pages 438–456. Springer, 2013.
 - [19] P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, and A. P. Sheth. Linked data is merely more data. In *AAAI Spring Symposium: linked data meets artificial intelligence*, volume 11, 2010.

- [20] K. Janowicz and P. Hitzler. Thoughts on the complex relation between linked data, semantic annotations, and ontologies. In *Proceedings of the sixth international workshop on Exploiting semantic annotations in information retrieval*, pages 41–44. ACM, 2013.
- [21] K. Janowicz, F. van Harmelen, J. A. Hendler, and P. Hitzler. Why the data train needs semantic rails. *AI Magazine*, 36(1):5–14, 2015.
- [22] A. G. Nuzzolese, A. Gangemi, V. Presutti, and P. Ciancarini. Encyclopedic knowledge patterns from Wikipedia links. In *International Semantic Web Conference*, pages 520–536. Springer, 2011.
- [23] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme design with content ontology design patterns. In *Proceedings of the 2009 International Conference on Ontology Patterns- Volume 516*, pages 83–97. CEUR-WS.org, 2009.
- [24] V. Presutti, F. Draicchio, and A. Gangemi. Knowledge extraction based on discourse representation theory and linguistic frames. In *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management*, EKAW’12, pages 114–129. Springer-Verlag, 2012.
- [25] V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Conceptual Modeling-ER 2008*, pages 128–141. Springer, 2008.
- [26] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 63–81. Springer, 2004.
- [27] V. Rodriguez-Doncel, A. A. Krisnadhi, P. Hitzler, M. Cheatham, N. Karima, and R. Amini. Pattern-based linked data publication: The linked chess dataset case. In *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, US*, 2015.
- [28] F. Scharffe, O. Zamazal, and D. Fensel. Ontology alignment design patterns. *Knowledge and information systems*, 40(1):1–28, 2014.
- [29] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *The semantic web-ISWC 2014*, pages 245–260. Springer, 2014.
- [30] J. Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly, April 2007.
- [31] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.
- [32] W. R. Van Hage, V. Malaisé, R. Segers, L. Hollink, and G. Schreiber. Design and use of the Simple Event Model (SEM). *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(2):128–136, 2011.
- [33] D. Vrandečić and A. Gangemi. Unit tests for ontologies. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, pages 1012–1020. Springer, 2006.
- [34] D. Vrandečić and Y. Sure. How to design better ontology metrics. In *European Semantic Web Conference*, pages 311–325. Springer, 2007.

Part I

Foundations of Ontology Design Patterns

This page intentionally left blank

Chapter 1

Modeling With Ontology Design Patterns: Chess Games As a Worked Example

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

1.1. Introduction

In this chapter, we provide a worked example for modeling with ontology design patterns (ODPs), as a way to set the stage for subsequent discussions. It should be noted, though, that there is not a single way how to model with ODPs effectively – the approach we follow here is mostly inspired by our own experiences with modeling at GeoVocamps [5, 9] and with scientists with diverse backgrounds [2, 3, 10, 11, 16, 20]. Another approach for designing ODPs, called the eXtreme Design Methodology, is discussed in another chapter of this book [1, 17]. Roughly, our approach here consists of (i) formulating use case(s) for which the ODP is intended; (ii) modeling key notions identified from the use case descriptions, first done visually, followed by writing the appropriate logical axiomatization; and (iii) checking the resulting ODP to ensure its quality, which can be done via checking the logical consistency of its axioms, inspecting some of its logical consequences (ensuring no unintended consequence is entailed), populating it with sample data to expose shortcomings in modeling, and testing the ODP against the use cases to ensure that they can be adequately answered by the ODP.

In practice, developing an ODP to model a notion in a particular domain typically requires a close collaboration with the domain experts. However, in this chapter, our modeling of “Chess Game” is helped by widely available chess game data in a number of chess game online repositories. Hence, we do not explicitly involve the domain experts.

We assume that the reader is familiar with basic knowledge about OWL and RDF, as conveyed in brief form in an appendix of this book [15], as well as with

Linked Data [4]. For a thorough introduction, see [6]. The chess example itself is based on the papers by the co-authors [13, 19].

1.2. Use Case

Chess is one of the most popular games worldwide, e.g., a 2012 study found that 605 million adults play chess regularly,¹ with millions playing online.² This significant interest in the game, and the large number of chess games played online, led to huge repositories of chess games that people can access in the form of Portable Game Notation (PGN) files. This chess game data does not just contain details about the moves of a game, but also about player identities, chess tournament names, and spatiotemporal information relevant to the games.

The use case that shall drive our modeling is that of enhanced searchability of chess games by potentially taking into account additional information from the Web, e.g. using player pages or pages on tournaments or openings on Wikipedia or on chess sites, using location information available through gazeteers or geo-related linked datasets, etc. This perspective precludes, for example, a straightforward conversion of PGN into linked data: In order to be able to merge information from other data sources, it will not suffice to represent players by strings consisting of names, as done in PGN.

Our modeling in fact shall be informed by the desire to remain as general as possible so that it would be possible, and any time later, to add information from additional sources, and such that we do not impose any significant restrictions on the parameters that can be used during the search. This means, it is a use case scenario where it is prudent to apply ontology design patterns and good ontology modeling techniques in order to arrive at an underlying schema which is robust, extendable, and easy to maintain and update, before creating the actual shared data as linked data based on this schema or ontology.

After settling on a use case, it is advisable to formulate a number of competency questions which shall be answerable over the data once it is published. Some typical competency questions are listed below.

- (i) Who played against Kasparov in the round 1994 Linares tournament? Did (s)he play as a white or black player?
- (ii) What is the first move taken by the black player in the Sicilian Defence opening?
- (iii) Find all games in which Bobby Fischer, playing black, lost in the poisoned pawn variation of the Sicilian Defence opening.
- (iv) Are there any recorded games using the Grünfeld Defence from before the 20th century?

¹<http://en.chessbase.com/post/che-redux-how-many-people-play-che->

²<https://www.chess.com/> lists over 13 million members as of February 2016, with tens of thousands concurrently online.

- (v) What did Kasparov say about his opponent's first two moves in his commentary about his game against Topalov in the 1999 Tournament in Wijk aan Zee?
- (vi) Who was the first non-Russian world champion after Fischer?
- (vii) Did Bobby Fischer ever play against a grandmaster in Germany?
- (viii) List all world championship games won by forfeit.

1.3. Data Sources and Scoping

The use case definition above of course already needs to be informed by some knowledge about available datasets that can be used for the intended purpose. Once we have a set of competency questions as listed above, we take a somewhat closer look at the possible data sources.

Central to our endeavor is the fact that there exists a de-facto standard for representing chess games, namely the above mentioned PGN format³ which is supported by many chess programs and online playing sites, and countless games in PGN format are available for download from the Web. Each game description in a PGN file contains a tag-pair section and a movetext section. The latter describes the actual moves in the game using the Standard Algebraic Notation (SAN), possibly with additional annotation, while the former provides information in the form of name-value pairs expressed as a tag enclosed with square brackets. The tag-pair section contains seven mandatory pieces information, namely the name of tournament or match event, the location of the event, the starting date of the game, the playing round for the game in a tournament, the white player's name, the black player's name, and the result of the game. Additionally, it may also contain other information such as players' Elo ratings, opening variation, alternative starting position, or description of the game conclusion. The PGN format thus makes it possible to record the most relevant information pertaining to a particular game of chess. Figure 1.1 contains a complete example of a simple PGN file.

PGN files will make it possible to address some of the competency questions, e.g., questions (i) through (iv), or even (v) if a PGN file with Kasparov's annotations on this game is available. However, PGN files alone will not make it possible to address competency questions like number (vi), which would require biographical information about players, (vii), which would require more fine-grained information on game locations, or (viii), which would require more fine-grained information about the cause for a game score, i.e., whether it was by check-mate, by resignation, or by forfeit.

Possible sources for such additional information are plentiful. Most convenient, of course, would be to incorporate sources that are already available as linked data, e.g., DBpedia [14] for additional information on players, openings, tournaments, or GeoNames⁴ for location-information to address questions such as

³http://www.thechessdrum.net/PGN_Reference.txt

⁴<http://www.geonames.org/ontology/documentation.html>

```
[Event "World Championship 31th-KK1"]
[Site "Moscow"]
[Date "1984.11.23"]
[Round "27"]
[White "Karpov, Anatoly"]
[Black "Kasparov, Gary"]
[Result "1-0"]
[WhiteElo "2705"]
[BlackElo "2715"]
[ECO "D55"]

1.Nf3 d5 2.d4 Nf6 3.c4 e6 4.Nc3 Be7 5.Bg5 h6 6.Bxf6 Bxf6 7.e3 0-0 8.Qc2 c5
9.dxc5 dxc4 10.Bxc4 Qa5 11.0-0 Bxc3 12.Qxc3 Qxc3 13.bxc3 Nd7 14.c6 bxc6
15.Rab1 Nb6 16.Be2 c5 17.Rfc1 Bb7 18.Kf1 Bd5 19.Rb5 Nd7 20.Ra5 Rfb8 21.c4 Bc6
22.Ne1 Rb4 23.Bd1 Rb7 24.f3 Rd8 25.Nd3 g5 26.Bb3 Kf8 27.Nxc5 Nxc5 28.Rxc5 Rd6
29.Ke2 Ke7 30.Rd1 Rxdi 31.Kxd1 Kd6 32.Ra5 f5 33.Ke2 h5 34.e4 fxe4 35.fxe4 Bxe4
36.Rxg5 Bf5 37.Ke3 h4 38.Kd4 e5+ 39.Kc3 Bb1 40.a3 Re7 41.Rg4 h3 42.g3 Re8
43.Rg7 Rf8 44.Rxa7 Rf2 45.Kb4 Rxh2 46.c5+ Kc6 47.Ba4+ Kd5 48.Rd7+ Ke4
49.c6 Rb2+ 50.Ka5 Rb8 51.c7 Rc8 52.Kb6 Ke3 53.Bc6 h2 54.g4 Rh8 55.Rd1 Ba2
56.Re1+ Kf4 57.Re4+ Kg3 58.Rxe5 Kxg4 59.Re2 1-0
```

Figure 1.1. Example PGN file.

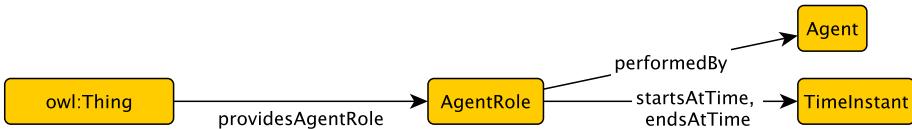
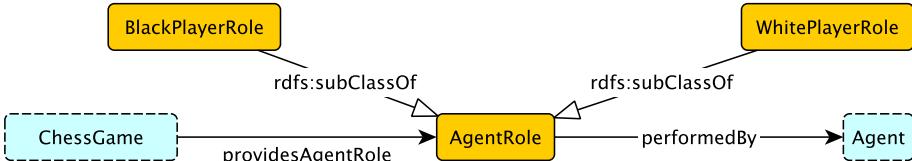
question (vii). Additional relevant data could e.g. be scraped from chess websites, to potentially further expand search capabilities for games.⁵

Of course, at this stage the scope, in terms of data coverage and search capabilities, needs to be considered. We will take a rather conservative perspective and intend to model essentially what is available from PGN files, in order to republish them as linked data. However at the same time we want to keep in mind that we may want the possibility to expand to other data sources later, i.e., our modeling should already reflect this, such that a broadening of scope is possible without the need to redo the modeling.

1.4. Modeling Key Notions

Since our main focus is on PGN files, it is tempting to let the ontology – and thus the shape of the resulting linked data RDF graph – be informed by the way PGN files encode information. However, this straightforward way of modeling would not account for the possible expansion of scope and thus required data integration which is part of our use case: We have already discussed problems arising, for example, out of identifying players by name-string only, as done in the PGN format. There is also another, perhaps less obvious reason to not go this route: modeling an ontology closely following a data format may make data publication simpler, but at the same time the resulting graph structures may end up being rather not intuitive, i.e., they may be far from capturing human conceptualization of this information, thus making the reuse by third parties much more difficult and costly [19]. We will thus follow a different approach: the PGN format as our

⁵Licensing issues of course need to be reflected [8, 18], but we will ignore this aspect since it is not directly relevant for our modeling example.

**Figure 1.2.** The Agent Role ODP.**Figure 1.3.** Adapted Agent Role ODP for chess player roles.

starting point will merely inform us of key notions that should be captured in our ontology. However, the structure of our ontology will be informed not by the PGN format, but rather by general ontology design patterns which much more faithfully capture human conceptualizations.

In this particular case, based on the required fields of the PGN format discussed earlier, and based on our competency questions, we can identify several key notions which will be in need of modeling, say *chess game*, *move* (or *half-move*⁶), *player*, *opening*, *result*, *commentary*, and *tournament*. In a next step, we look at each of these key notions in turn, and attempt to determine what general pattern it should be modeled after.

Let us start with the notion of *player*. In order to be able to attach additional information to a player, such as the player's ELO score or citizenship information, we need of course a dereferenceable URI. However, in addition to this, we want to borrow from best practices and realize that "being a player" is actually a *role* which an agent can take, e.g., for a certain time period. We thus reuse the very common ontology design pattern for this purpose, which is depicted in Figure 1.2 in exactly the form in which it is used, e.g., in [11]. In this ODP, something (e.g., a chess game) provides roles (such as white player or referee), which an agent (e.g., Magnus Carlsen or Deep Blue) holds for some time interval.

For the chess game, we want to adapt this pattern for our specific case. We prefer to leave most things untouched, while avoiding overgeneralization (e.g., we want to get rid of the owl:Thing type) and while pruning unnecessary details. The resulting pattern is depicted in Figure 1.3. Note in particular that we have removed the temporal extent, which should rather be attached to the chess game itself. We also introduced two subclasses to distinguish the two different player roles, white player and black player. The dashed frames indicate that a more complex entity (a pattern in its own right) could stand in place of the frame:

⁶In chess, one player's turn is considered a *half-move*, while a *move* consists of two half-moves: one by the white player followed by one by the black player.

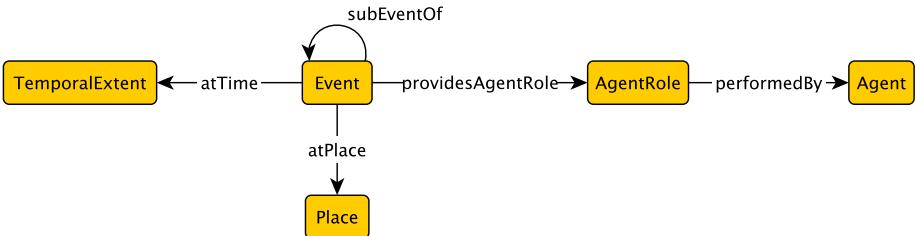


Figure 1.4. The Event ODP.

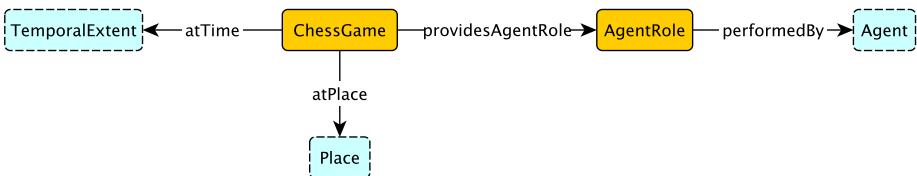


Figure 1.5. Chess game as event.

ChessGame will indeed be modeled below, and Agent could also be expanded, if desired, with a more fine-grained model; or alternatively, an existing Agent ODP could be directly used instead.

In our experience, agent roles occur very often when modeling, i.e. the Agent Role ODP is particularly useful at many occasions.

We now turn our attention to *chess game*. There are different possible perspectives from which one could understand the notion of chess game. It could, e.g., be understood as an abstract artifact similar to a piece of art, with the players being the creators. One could also view it as an event, or alternatively as a record of an event. Each of these perspectives has merit and it is not always straightforward to find the most suitable one; in tricky cases it is in our experience helpful to have a group of people discuss the different viewpoints in light of the competency questions and the use case.

In our specific case, our competency questions indicate that time and space related to a chess game play an identifying role. Furthermore, agents take particular roles (e.g., as players) relative to a chess game. These observations suggest to reuse a generic and rather straightforward *event* pattern, such as the one depicted in Figure 1.4, which is a variant of the form used by the co-authors in the past [11]. It essentially lists only spatial and temporal extent, indicates that events may provide agent roles, and allows an event to have a sub-event.

Like before, we specialize the pattern for our specific purpose, the result of which is depicted in Figure 1.5. Of course we reuse the Agent Role pattern. We do at this stage not address the question how exactly we want to represent temporal and spatial information; others have indeed already provided solutions to this, e.g., the OWL Time Ontology [7]. Moreover, the subevent relationship is not employed at this stage.

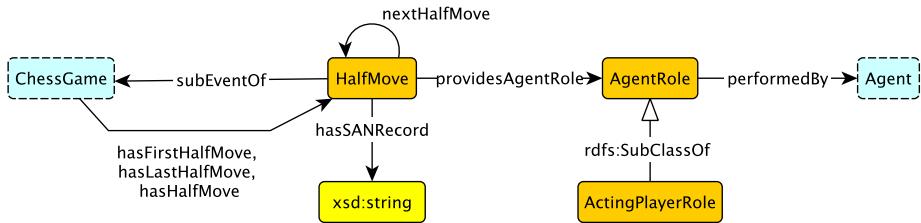
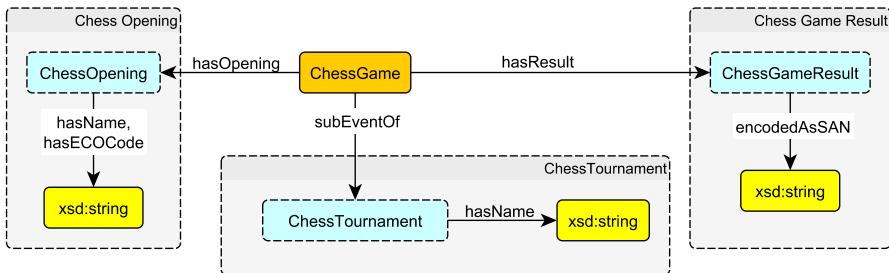


Figure 1.6. Half-moves.

Figure 1.7. Three *stubs*, for ChessOpening, ChessTournament, and ChessGameResult.

Let us next look at *move*, or rather at *half-move* which is the technical term in chess for one player's action. Since we have settled for viewing chess games as events, half-moves are naturally subevents. This also aligns with the idea that half-moves may have their own temporal extent, which may be recorded (and very relevant for the game outcome) by means of the chess clock used during the game. Half-moves provide an agent role, namely that of acting player, and each half-move has a record, which can be given in Standard Algebraic Notation (SAN),⁷ the undisputed norm for recording half-moves which is also incorporated in the PGN format. Each half-move furthermore refers to the next half-move, so that the sequence is conveyed,⁸ and we need to identify first and last half-moves of the game.⁹ The resulting pattern is depicted in Figure 1.6.

Let us next look at *tournament*. Some of our competency questions indicate that we may eventually want a fine-grained modeling of chess tournaments. However, let us say that we agree to not model this aspect in detail at this stage, but that we rather intend to provide the possibility to make this extension later, without having to change anything already modeled.

At this stage, say we want only the ability to record the name of a tournament as a string. For extendability we also require a *hook*, in form of the central node for a chess tournament, to which additional information can later be attached.

⁷<http://www.fide.com/component/handbook/?id=171&view=article> Appendix C

⁸We can also understand this as a derivative of a generic *sequence* ODP of course.

⁹Some of this, e.g. which player's turn it is, is really redundant information, but it seems convenient to nevertheless record it explicitly.

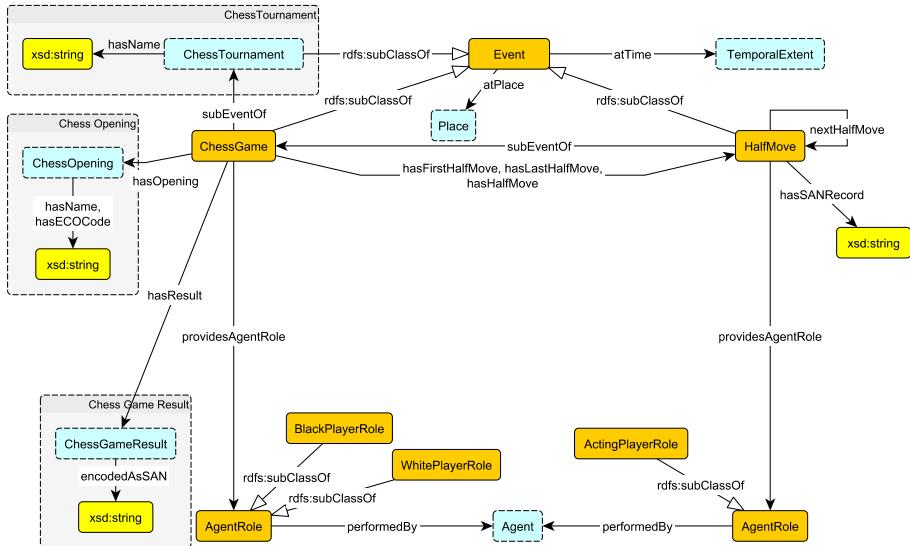


Figure 1.8. Putting the previously developed pieces together. Orange boxes are atomic classes. Blue boxes with dashed frames are classes with details to be developed or hidden in a separate pattern/ontology. Grouping boxes are sub-patterns that can be modeled separately.

This can be realized best with a *stub*, as depicted in the lower part of Figure 1.7. Note that this way of modeling chess tournaments is rather uncommittal regarding future extensions. In contrast, if we had directly linked chess games to the string containing the name of the tournament, we would not have provided an obvious hook for the future extension.

Let us currently decide to do the same with *opening*, however adding a way to record the ECO (Encyclopedia of Chess Openings) code for the opening. And let's also do the same for now with the result of a chess game, recorded using SAN, and again this could be extended later to add the ability to answer competency questions like number viii.

The only key notion missing from our initial list is *commentary*, but before we do this, it seems time to start putting things together.

1.5. Putting Things Together

It is straightforward to assemble the pieces we already developed, and the result is depicted in Figure 1.8. Note that we attached the temporal and spatial extents to *Event*: due to the subclassing of *ChessGame* (and *ChessTournament*) this means that temporal and spatial extents can also be specified for these. The picture shows two nodes for *AgentRole* simply because it makes the graph look nicer, and because it is then easier to see which agent roles are available in each case.

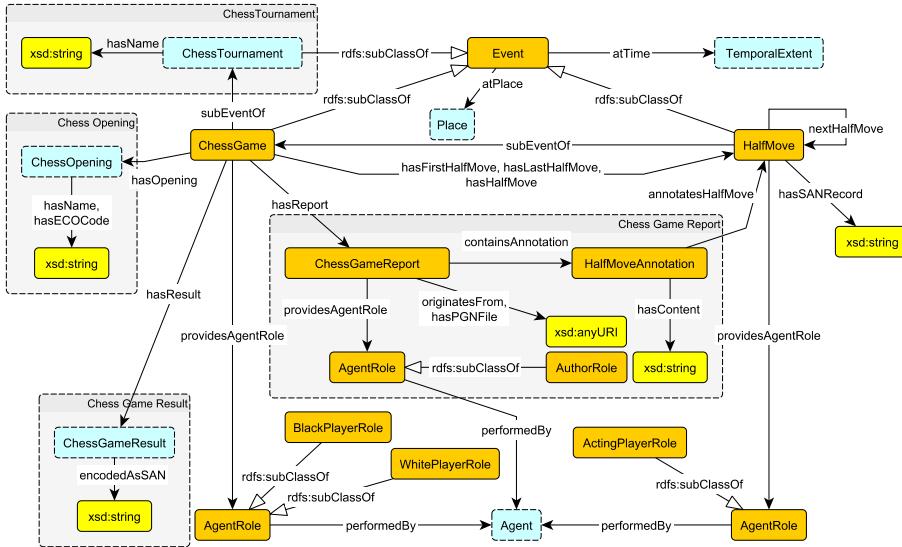


Figure 1.9. Depiction of the complete model.

Note also that the previously developed pieces can essentially be understood as a type of *modules* [11], each derived from ODPs, which are put together to form the ontology outline.

Now let's go back to *commentary*, which is arguably a bit more tricky to deal with. A commentary in chess usually consists of a set of comments, each of which is attached to a half-move. A comment thus comes attached to the move sequence, e.g. within a PGN file. We thus may have different commentaries for one and the same game, just as we may have different PGN files for the same game, which may or may not differ in some respects. There are different ways how this can be viewed. One could consider the PGN files to be *manifestations* of a game – in the sense in which, say, two different hardcopies of a book are two concrete manifestations of one and the same book. This perspective was taken in [13, 19]. A different perspective would be to consider the PGNs, and also commentaries, as *reports* on the chess game events. We will take this latter perspective herein.

Once we have settled on this perspective, the remaining pieces fall into place relatively easily. Reports provide author roles (modeled using the Agent Role ODP), and carry Annotations (i.e., comments) which are attached to half-moves and have strings as content. We furthermore want to record that a report may originate from some URI, or has an associated PGN file we would want to point to. The resulting ontology (we could also call it *Chess Game ODP*) is depicted in Figure 1.9.

1.6. Axiomatization

The diagram of course is not the ontology, it is merely a preliminary depiction of main relationships, which is appropriate at this stage because so far the discussion has not dived into details. We have so far avoided issues such as specifying that a chess game can only have one first half move, for example, and while they may be intuitively understood by a human reader familiar with chess, they may not be so clear for others, and in any case we would like to put as many of these conditions into the formal model as possible.

For this purpose, we describe the pattern using *axioms* in some logic. Below, we use description logics, which in this case can be translated seamlessly into OWL. From the earlier sections, we know that the chess game pattern consists of the following components:

1. the (re)use of the Agent Role pattern, which reoccurs when modeling players and authors of chess game reports;
2. the (re)use of the Event pattern to model chess game, half move, and chess tournament;
3. stubs for representing chess opening and chess game result; and
4. simple pattern for chess game report.

One way to realize the reuse of Agent Role and Event patterns in practice is by importing the OWL serialization of both patterns into the OWL serialization of the Chess Game pattern. This means that all axioms and ontological commitments imposed by both Agent Role and Event patterns will be employed by the Chess Game pattern. In general, however, the use cases and modeling requirements may necessitate some adjustments and modifications in the axioms during pattern reuse, which cannot be done solely through OWL import statements. To simplify our discussion in this chapter, the Agent Role and Event patterns are modeled so that it is not necessary to modify them when importing into the Chess Game pattern. We begin with the Agent Role pattern, follow it with the Event pattern, and conclude the discussion with the Chess Game pattern itself.

1.6.1. Agent Role

Axioms for the Agent Role pattern are depicted in Figure 1.2. The axioms for Agent Role pattern are given in Figure 1.10. Further discussion on the Agent Role pattern can be found in Chapter 16. In the meantime, we provide a brief explanation of the axioms here.

Axiom (1.1) and (1.2) asserts that every agent role is always performed by exactly one agent, starts at exactly one time instant, and ends at exactly one time instant. Note that these axioms only states the existence of an agent, starting time instant, and ending time instant for an agent role; it does not imply that they have to be *known* nor they have to be included in the data.

The other axioms in the pattern, except the last one, capture the domain and range of each property in the pattern. Roughly, for every triple of class *A*, (object

$$\begin{aligned}
 \text{AgentRole} &\sqsubseteq (=1 \text{ performedBy}.\text{Agent}) & (1.1) \\
 \text{AgentRole} &\sqsubseteq (=1 \text{ startsAtTime}.\text{TimeInstant}) \sqcap (=1 \text{ endsAtTime}.\text{TimeInstant}) & (1.2) \\
 \exists \text{performedBy}.\text{Agent} &\sqsubseteq \text{AgentRole} & (1.3) \\
 \exists \text{startsAtTime}.\text{TimeInstant} \sqcup \exists \text{endsAtTime}.\text{TimeInstant} &\sqsubseteq \text{AgentRole} & (1.4) \\
 \text{AgentRole} &\sqsubseteq \forall \text{performedBy}.\text{Agent} & (1.5) \\
 \text{AgentRole} &\sqsubseteq \forall \text{startsAtTime}.\text{TimeInstant} \sqcap \forall \text{endsAtTime}.\text{TimeInstant} & (1.6) \\
 \top &\sqsubseteq \forall \text{providesAgentRole}.\text{AgentRole} & (1.7) \\
 \text{DisjointClasses}(\text{AgentRole}, \text{Agent}, \text{TimeInstant}) & & (1.8)
 \end{aligned}$$

Figure 1.10. Axiomatization for Agent Role ODP

or data) property P , and class or data type B , we want to express that the domain of P is A and its range is B . We achieve this using the so-called scoped domain and range restrictions, which we shall mostly follow throughout this chapter. We begin with axiom (1.3), which states that if something is performed by an agent, then it is an agent role. This is a form of scoped domain restriction for the property `performedBy` where the domain is `AgentRole` and the scope is `Agent`. More precisely, given two individuals x and y such that x is performed by y , we infer that x is an agent role (i.e., belongs to the domain) if y is an agent (i.e., is known to belong to the scope). If y is not known to be an agent, then we do not infer that x is an agent role. This differs from directly asserting that the domain of `performedBy` is `AgentRole`. The latter means that if x is performed by y , then x is an agent role regardless whether y is an agent or not. Axiom (1.4) is likewise scoped domain restriction for both `startsAtTime` and `endsAtTime`.¹⁰

Axiom (1.5) is a scoped range restriction for the property `performedBy` with `Agent` as the range and `AgentRole` as the corresponding scope. This axiom states that for two individuals x and y , given two individuals x and y such that x is performed by y , if x is an agent role, then we infer that y is an agent. If x is not known to be an agent role, we do not infer y to be an agent here. Axiom (1.6) expresses the scoped range restriction for both `startsAtTime` and `endsAtTime` property.¹¹.

Next, anything may provide an agent role, i.e., it provides zero or more agent roles. For this requirement, we do not write an axiom involving a number restriction because for every property R and class C , $(\geq 0 R.C)$ is equivalent to \top . Instead, we simply express the domain and range restrictions for the property. Specific for the Agent Role pattern, the domain of `providesAgentRole` property is `owl:Thing`, so there is no need to explicitly state a domain restriction for this property, and we simply assert the (unscoped) range restriction according to axiom (1.7).

¹⁰ $C \sqcup D \sqsubseteq E$ is equivalent to two axioms $C \sqsubseteq E$ and $D \sqsubseteq E$, as an easy consequence of the OWL 2 semantics.

¹¹Again, OWL 2 semantics implies that $C \sqsubseteq D \sqcap E$ is equivalent to two axioms $C \sqsubseteq D$ and $C \sqsubseteq E$.

import: (1.1), (1.2), (1.3), (1.4), (1.5), (1.6), (1.7), (1.8)

$\text{Event} \sqsubseteq \exists \text{atPlace}.\text{Place} \sqcap \exists \text{atTime}.\text{TemporalExtent}$ (1.9)

$\exists \text{atTime}.\text{TemporalExtent} \sqcup \exists \text{atPlace}.\text{Place} \sqcup \exists \text{subEventOf}.\text{Event} \sqsubseteq \text{Event}$ (1.10)

$\text{Event} \sqsubseteq \forall \text{atTime}.\text{TemporalExtent} \sqcap \forall \text{atPlace}.\text{Place} \sqcap \forall \text{subEventOf}.\text{Event}$ (1.11)

$\text{DisjointClasses}(\text{Event}, \text{TemporalExtent}, \text{Place}, \text{AgentRole}, \text{Agent})$ (1.12)

Figure 1.11. Axiomatization for Event ODP

Finally, axiom (1.8) asserts that every pair of classes amongst `AgentRole`, `Agent`, and `TimeInstant` are pairwise disjoint. That is, for example, nothing can be simultaneously an agent and an agent role, or nothing can be both an agent role and a time instant.

1.6.2. Event

Axioms for the Event pattern are depicted in Figure 1.11. Note that the Event pattern imports the Agent Role pattern.

Axiom (1.9) asserts that an event occurs at some temporal extent and at some place. Again, the temporal extent and place may not necessarily be known.

Axiom (1.10) expresses scoped domain restrictions for the property `atTime`, `atPlace`, and `subEventOf` where the domain of all three properties is `Event` while the scope are `TemporalExtent`, `Place`, and `Event`, respectively. Meanwhile, axiom (1.11) expresses the corresponding scoped range restrictions where the range are respectively `TemporalExtent`, `Place`, and `Event`, while the class `Event` is the scope for range restrictions for all those three properties. Since the Agent Role pattern is imported to the Event pattern, we directly use their axioms and do not repeat their assertion here, including domain and range restrictions for `providesAgentRole` and `performedBy` property.

Finally, axiom (1.12) asserts any two different classes amongst `Event`, `TemporalExtent`, `Place`, `AgentRole`, and `Agent` are pairwise disjoint.

1.6.3. Chess Game

Axioms for Chess Game pattern are given in Figure 1.12 and 1.13. Note that axioms from the Agent Role and Event pattern are imported into the Chess Game pattern.

Axiom (1.13) asserts that a chess game is an event and it provides a black player role and a white player role. Since every chess game is an event, axioms for event apply and thus, a chess game happens at some time and some place. Next, every black player role and white player role themselves are agent roles, and furthermore, there exists exactly one chess game that provides them. This was asserted in axiom (1.14).

import: (1.1), (1.2), (1.3), (1.4), (1.5), (1.6), (1.7), (1.8), (1.9), (1.10), (1.11), (1.12)

$$\text{ChessGame} \sqsubseteq \text{Event} \sqcap \exists \text{pAR.BlackPlayerRole} \sqcap \exists \text{pAR.WhitePlayerRole} \quad (1.13)$$

$$\text{BlackPlayerRole} \sqcup \text{WhitePlayerRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^-. \text{ChessGame}) \quad (1.14)$$

$$\text{ChessGame} \sqsubseteq (=1 \text{ hasFirstHalfMove.} \text{HalfMove}) \quad (1.15)$$

$$\text{ChessGame} \sqsubseteq (=1 \text{ hasLastHalfMove.} \text{HalfMove}) \quad (1.16)$$

$$\text{hasHalfMove} \sqsubseteq \text{subEventOf}^- \quad (1.17)$$

$$\text{hasFirstHalfMove} \sqsubseteq \text{hasHalfMove} \quad (1.18)$$

$$\text{hasLastHalfMove} \sqsubseteq \text{hasHalfMove} \quad (1.19)$$

$$\exists \text{subEventOf.} \text{ChessTournament} \sqcup \exists \text{hasOpening.} \text{ChessOpening} \sqsubseteq \text{ChessGame} \quad (1.20)$$

$$\exists \text{hasResult.} \text{ChessGameResult} \sqcup \exists \text{hasReport.} \text{ChessGameReport} \sqsubseteq \text{ChessGame} \quad (1.21)$$

$$\text{ChessGame} \sqsubseteq \forall \text{subEventOf.} \text{ChessTournament} \sqcap \forall \text{hasOpening.} \text{ChessOpening} \quad (1.22)$$

$$\text{ChessGame} \sqsubseteq \forall \text{hasResult.} \text{ChessGameResult} \sqcap \forall \text{hasReport.} \text{ChessGameReport} \quad (1.23)$$

$$\text{HalfMove} \sqsubseteq \text{Event} \sqcap \exists \text{pAR.ActingPlayerRole} \sqcap (=1 \text{ hasHalfMove}^- \text{.} \text{ChessGame}) \quad (1.24)$$

$$\text{ActingPlayerRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^-. \text{HalfMove}) \quad (1.25)$$

$$\text{HalfMove} \sqsubseteq (\leq 1 \text{ nextHalfMove.} \text{HalfMove}) \sqcap \neg \exists \text{nextHalfMove.} \text{Self} \quad (1.26)$$

$$\exists \text{subEventOf.} \text{ChessGame} \sqcup \exists \text{nextHalfMove.} \text{HalfMove} \sqsubseteq \text{HalfMove} \quad (1.27)$$

$$\exists \text{hasSANRecord.xsd:string} \sqsubseteq \text{HalfMove} \quad (1.28)$$

$$\text{HalfMove} \sqsubseteq \forall \text{subEventOf.} \text{ChessGame} \sqcap \forall \text{nextHalfMove.} \text{HalfMove} \quad (1.29)$$

$$\text{HalfMove} \sqsubseteq \forall \text{hasSANRecord.xsd:string} \quad (1.30)$$

Figure 1.12. Chess Game pattern (part 1) with some axioms imported from the Agent Role (Fig. 1.10) and Event pattern (Fig. 1.11); pAR stands for providesAgentRole. The rest of the axioms can be found in Fig. 1.13

Axiom (1.15) and (1.16) state that every chess game has exactly one half move in the beginning (first half move) and one at the end (last half move). Axiom (1.17) indicates that the relation between half-move and chess game is that of a sub-event. This is important in case one may want to impose further axioms coming from an event pattern, e.g., one could specify that sub-events of an event must fall within the spatial and temporal scope of the larger event. We have not done this here. Furthermore, axiom (1.18) and (1.19) assert that first half-move and last half-move are actually half-moves.

Next, a chess game may be a sub-event of a chess tournament, may have a chess opening, and may have a chess game result, and may have a chess game report. The reader may notice that stating that a chess game may be a sub-event of a chess tournament is actually equivalent to saying that a chess game is a sub-event of zero or more chess tournaments. This can be modeled as the axiom $\text{ChessGame} \sqsubseteq (\geq 0 \text{ subEventOf.} \text{ChessTournament})$. As discussed earlier when discussing the Agent Role pattern, the right-hand side of this axiom is equivalent to \top , hence the axiom is equivalent to asserting $\text{ChessGame} \sqsubseteq \top$, which gives us

$\text{ChessTournament} \sqsubseteq \text{Event}$	(1.31)
$\text{ChessTournament} \sqcup \text{ChessOpening} \sqsubseteq \forall \text{hasName.xsd:string}$	(1.32)
$\exists \text{hasECOCode.xsd:string} \sqsubseteq \text{ChessOpening}$	(1.33)
$\text{ChessOpening} \sqsubseteq \forall \text{hasECOCode.xsd:string}$	(1.34)
$\exists \text{encodedAsSAN.xsd:string} \sqsubseteq \text{ChessGameResult}$	(1.35)
$\text{ChessGameResult} \sqsubseteq \forall \text{encodedAsSAN.xsd:string}$	(1.36)
$\text{ChessGameReport} \sqsubseteq \exists pAR.\text{AuthorRole}$	(1.37)
$\text{AuthorRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^{-}.\text{ChessGameReport})$	(1.38)
$\exists \text{containsAnnotation.HalfMoveAnnotation} \sqsubseteq \text{ChessGameReport}$	(1.39)
$\text{ChessGameReport} \sqsubseteq \forall \text{containsAnnotation.HalfMoveAnnotation}$	(1.40)
$\exists \text{originatesFrom.xsd:anyURI} \sqcup \exists \text{hasPGNFile.xsd:anyURI} \sqsubseteq \text{ChessGameReport}$	(1.41)
$\text{ChessGameReport} \sqsubseteq \forall \text{originatesFrom.xsd:anyURI} \sqcap \forall \text{hasPGNFile.xsd:anyURI}$	(1.42)
$\text{HalfMoveAnnotation} \sqsubseteq (=1 \text{ annotatesHalfMove.HalfMove})$	(1.43)
$\text{HalfMoveAnnotation} \sqsubseteq \exists \text{hasContent.xsd:string}$	(1.44)
$\exists \text{hasContent.xsd:string} \sqsubseteq \text{HalfMoveAnnotation}$	(1.45)
$\text{HalfMoveAnnotation} \sqsubseteq \forall \text{hasContent.xsd:string}$	(1.46)
$\text{DisjointClasses}(\text{AgentRole}, \text{Agent}, \text{Event}, \text{Place}, \text{TemporalExtent},$	
$\text{ChessOpening}, \text{ChessGameResult}, \text{ChessGameReport},$	(1.47)
$\text{HalfMoveAnnotation})$	
$\text{DisjointClasses}(\text{ChessGame}, \text{ChessTournament}, \text{HalfMove})$	(1.48)
$\text{DisjointClasses}(\text{BlackPlayerRole}, \text{WhitePlayerRole}, \text{ActingPlayerRole},$	
$\text{AuthorRole})$	(1.49)

Figure 1.13. Chess Game pattern (part 2; continued from Fig. 1.12); pAR stands for providesAgentRole.

nothing in terms of inference because it always holds by definition of the semantics of OWL. We thus do not include this in the axiomatization. The same thing holds for the relationships with chess opening, chess game result, and chess game report.

Nevertheless, the corresponding object and data properties that capture such relationships shall still be included in the axiomatization in the form of domain and range restrictions. Axiom (1.20) and (1.21) simultaneously express scoped domain restrictions for subEventOf, hasOpening, hasResult, and hasReport. The class ChessGame is the domain of all four properties, while the scope are respectively ChessTournament, ChessOpening, ChessGameResult, and ChessGameReport.

Axiom (1.22) and (1.23) assert scoped range restrictions for subEventOf, hasOpening, hasResult, and hasReport. Here, the range are ChessTournament, ChessOpening, ChessGameResult, and ChessGameReport, respectively, while the

class `ChessGame` acts as the scope for range restrictions of those four properties.

Axiom (1.24) states that every half-move is an event that provides an acting player role and furthermore, it must be a half-move of exactly one chess game. Here, an acting player role refers to the role of the chess player that performs the half-move. Also, axiom (1.25) expresses that an acting player role is an agent role, which is provided by exactly one half-move.

Axiom (1.26) asserts that a half-move cannot be followed by more than one half-move and cannot precede itself. This ensures that using the property `nextHalfMove`, we obtain a linear chain of half-moves.

Axiom (1.27) simultaneously expresses scoped domain restrictions for the property `subEventOf` and `nextHalfMove` where the domain is `HalfMove` for both properties and the scope are `ChessGame` and `HalfMove`, respectively. Notice that axiom (1.27) is a different domain restriction for `subEventOf` than the one given in axiom (1.20): the former asserts the domain of `subEventOf` to be the class `HalfMove`, provided that the scope is `ChessGame`, whereas the latter asserts the domain of `subEventOf` to be the class `ChessGame` if the scope is `ChessTournament`. These two restrictions can be asserted together because we use scoped domain restriction, i.e., this cannot be achieved using only non-scoped domain restrictions. In addition, axiom (1.28) is a scoped domain restriction for the data property `hasSANRecord` with `HalfMove` as the domain and `xsd:string` as the scope.

Axiom (1.29) asserts scoped range restrictions for the property `subEventOf` and `nextHalfMove` simultaneously where the scope is `HalfMove` for both properties and the range are `ChessGame` and `HalfMove`, respectively. Similar to the case for scoped domain restrictions explained earlier, this axiom asserts the second range restriction for `subEventOf`. Also, axiom (1.30) is a scoped range restriction for the data property `hasSANRecord` with `xsd:string` as the range and `HalfMove` as the scope.

Next, chess tournaments are events, as asserted in axiom (1.31). Axiom (1.32) asserts a scoped range restriction for the data property `hasName` with `xsd:string` as the range and the class union `ChessTournament` \sqcup `ChessOpening` as the scope. Next, axiom (1.33) asserts a scoped domain restriction for `hasECOCode` with `ChessOpening` as the domain and `xsd:string` as the scope, while axiom (1.34) expresses a scoped range restriction for the same property with `xsd:string` as the range and the class `ChessOpening` as the scope. Axiom (1.35) is a scoped domain restriction for `encodedAsSAN` with `ChessGameResult` as the domain and `xsd:string` as the scope, and axiom (1.36) is a scoped range restriction for the same property with `xsd:string` as the range and `ChessGameResult` as the scope.

The next few axioms are intended for the chess game report part of the pattern. Recall that the relationship between chess games and chess game reports is provided by the property `hasReport` whose domain and range were axiomatized in axiom (1.21) and (1.23). Now, a chess game report always provides at least one author role as given by axiom (1.37). Meanwhile, an author role is an agent role and it is provided by exactly one chess game report as asserted by axiom (1.38).

Axiom (1.39) asserts a scoped domain restriction for `containsAnnotation` property with `ChessGameReport` as the domain and `HalfMoveAnnotation` as the scope. Axiom (1.40) asserts a scoped range restriction for the same property with `HalfMoveAnnotation` as the range and `ChessGameReport` as the scope.

Axiom (1.41) asserts scoped domain restrictions for both `originatesFrom` and `hasPGNFile`, respectively. Here, the domain for both is `ChessGameReport` while the scope for both is `xsd:anyURI`. Axiom (1.42) asserts scoped range restrictions for both data properties with `xsd:anyURI` as the range and `ChessGameReport` as the scope for both.

Axiom (1.43) states that every half-move annotation annotates exactly one half-move. Axiom (1.44) asserts that every half-move annotation must have some string as content. Axiom (1.46) expresses a scoped domain restriction with `HalfMoveAnnotation` as the domain and `xsd:string` as the scope, while axiom (1.46) expresses a scoped range restriction for `hasContent` with `xsd:string` as the range and `HalfMoveAnnotation` as the scope.

Finally, axiom (1.47), (1.48), and (1.49) indicate pairwise disjointness between classes. The first one asserts that any two classes from `AgentRole`, `Agent`, `Event`, `Place`, `TemporalExtent`, `ChessOpening`, `ChessGameResult`, `ChessGameReport`, and `HalfMoveAnnotation` are pairwise disjoint. The second asserts pairwise disjointness between subclasses of `Event`, while the third asserts pairwise disjointness between subclasses of `AgentRole` in the Chess Game pattern.

1.7. Final Assessment

Quality assurance regarding the model can now be done the usual way, e.g., it would be prudent to check the set of axioms for consistency, and compute some logical consequences and inspect them as a type of sanity check. And of course, the model should be populated using real data, which sometimes exposes shortcomings. An example of a step-by-step process to populate the model with linked data can be found in a separate chapter of this book [12].

We will wrap up this chapter by returning to the competency questions stated earlier. Question (i) is answerable if the tournament can be identified, which may even be possible with the stub given. Question (ii) again is answerable by identifying the opening using name or ECO code, and then navigating to the first black half-move. Question (iii) again is identifiable e.g. by using name or ECO code for this variation. Question (iv) is possible by looking at the temporal extent given for games or their associated tournaments. Question (v) is answerable if the comment has been captured in some of the source data. Question (vi) is currently not answerable, but could be answerable by an appropriate extension of the Agent pattern in the model, which could, e.g., encompass nationalities. Question (vii) is answerable if player roles be enhanced by stating grandmaster status, and if enhanced location information is provided for tournaments and games. Question (viii) is currently not answerable, but our game result stub provides a hook for indicating whether a win was by forfeit, i.e. our model can also be extended in this direction.

Acknowledgements. This work was partially supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*.

Bibliography

- [1] E. Blomqvist, K. Hammar, and V. Presutti. Engineering ontologies with patterns – the eXtreme design methodology. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [2] D. Carral. An ontology design pattern for detector final states. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [3] D. Carral, M. Cheatham, S. Dallmeier-Tiessen, P. Herterich, M. D. Hildreth, P. Hitzler, A. Krisnadhi, K. Lassila-Perini, E. Sexton-Kennedy, C. Vardeman, and G. Watts. An ontology design pattern for particle physics analysis. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [4] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.
- [5] P. Hitzler, K. Janowicz, and A. A. Krisnadhi. Ontology modeling with domain experts: The GeoVocamp experience. In C. d'Amato, F. Lécué, R. Mutharaju, T. Narock, and F. Wirth, editors, *Proceedings of the 1st International Diversity++ Workshop co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015.*, volume 1501 of *CEUR Workshop Proceedings*, pages 31–36. CEUR-WS.org, 2015.
- [6] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.
- [7] J. R. Hobbs and F. Pan, editors. *Time Ontology in OWL*. W3C Working Draft, 27 September 2006. Available from <http://www.w3.org/TR/owl-time>.
- [8] P. Jain, P. Hitzler, K. Janowicz, and C. Venkatramani. There's no money in linked data. Available from <http://daselab.org/sites/default/files/No-Money-LOD-TechReport2013.pdf>, May 2013.
- [9] K. Janowicz. Modeling ontology design patterns with domain experts – a view from the trenches. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [10] A. Krisnadhi. *Ontology Pattern-Based Data Integration*. PhD thesis, Wright State University, 2015.
- [11] A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. A. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. W. Finin, P. Ji, M. B. Jones, N. Karima, K. Lehnert, A. Mickle, T. W. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The GeoLink Mod-

- ular Oceanography Ontology. In M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11–15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 2015.
- [12] A. Krisnadhi, N. Karima, P. Hitzler, R. Amini, V. Rodríguez-Doncel, and K. Janowicz. Ontology design patterns for linked data publishing. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
 - [13] A. Krisnadhi, V. Rodríguez-Doncel, P. Hitzler, M. Cheatham, N. Karima, R. Amini, and A. Coleman. An ontology design pattern for chess games. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
 - [14] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
 - [15] F. Maier. A primer on RDF and OWL. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
 - [16] P. O'Brien, D. Carral, J. Mixter, and P. Hitzler. An ontology design pattern for data integration in the library domain. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
 - [17] V. Presutti et al. eXtreme Design with content ontology design patterns. In E. Blomqvist et al., editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009), Washington D.C., USA, 25 October 2009.*, volume 516 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
 - [18] V. Rodríguez-Doncel, A. Gómez-Pérez, and N. Mihindukulasooriya. Rights declaration in linked data. In O. Hartig, J. Sequeda, A. Hogan, and T. Matsutsuka, editors, *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*, volume 1034 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
 - [19] V. Rodríguez-Doncel, A. A. Krisnadhi, P. Hitzler, M. Cheatham, N. Karima, and R. Amini. Pattern-based linked data publication: The linked chess dataset case. In O. Hartig, J. Sequeda, and A. Hogan, editors, *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11–15, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

Pennsylvania, US, October 12th, 2015., volume 1426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

- [20] B. Yan, Y. Hu, B. Kuczenski, K. Janowicz, A. Ballatore, A. A. Krisnadhi, Y. Ju, P. Hitzler, S. Suh, and W. Ingwersen. An ontology for specifying spatiotemporal scopes in life cycle assessment. In C. d'Amato, F. Lécué, R. Mutharaju, T. Narock, and F. Wirth, editors, *Proceedings of the 1st International Diversity++ Workshop co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015.*, volume 1501 of *CEUR Workshop Proceedings*, pages 25–30. CEUR-WS.org, 2015.

This page intentionally left blank

Chapter 2

Engineering Ontologies with Patterns – The eXtreme Design Methodology

Eva Blomqvist, Linköping University, Sweden

Karl Hammar, Jönköping University, Sweden

Valentina Presutti, ISTC-CNR, Italy

When using Ontology Design Patterns (ODPs) for modelling new parts of an ontology, i.e., new ontology modules, or even an entire ontology from scratch, ODPs can be used both as inspiration for different modelling solutions, as well as concrete templates or even “building blocks” reused directly in the new solution. This chapter discusses how ODPs, and in particular Content ODPs¹, can be used in ontology engineering. In particular, a specific ontology engineering methodology is presented, which was originally developed for supporting ODP use. However, this methodology, the eXtreme Design (XD), also has some characteristics that set it apart from most other ontology engineering methodologies, and which may be interesting to consider regardless of how much emphasis is put on the ODP usage. Towards the end of the chapter some XD use cases are also reported and discussed, as well as lessons learned from applying XD. The chapter is concluded through a summary and discussion about future work.

2.1. ODP-based Modelling – Different Aspects of ODP Use

ODPs were originally proposed partly as a result of observing how difficult it is to reuse a large ontology. This observation even includes foundational ontologies clearly designed for being reused as the basis for building other ontologies. Issues include that it is difficult to get an overview of such large ontologies, foresee effects of changes or extensions to them, and it is also rarely the case that you as an ontology engineer, or the set of requirements you have for your ontology

¹In fact, throughout this chapter when mentioning ODPs, this mainly refers to Content ODPs if not specified further.

engineering task at hand, will fully agree with all the ontological commitments that are made in such a large ontology. However, not reusing any well-established practices at all, and not aligning yourself at least partly to existing ontologies, will create problems in interoperability and potentially also understandability of your ontology. Hence, there is a trade-off between interoperability on one hand and overcommitment and conflicting requirements on the other hand, where ODPs as small “building blocks” offer one way to manage this trade-off. This is true regardless of whether reuse is discussed in a global Semantic Web context, or internally in an organisation, for instance. Hence, the idea of reusing, applying and sharing small patterns instead of complete ontologies, applies in many contexts. This is why, later when the XD methodology is discussed, it is mentioned that an important step is to decide what ODPs to use and how to create and manage your own ODPs, rather than there being a universally agreed set of ODPs that every project should use.

There are also many different types of ODPs, and they can be reused and applied in many different ways. Even when considering only what is called Content ODPs, i.e., ODPs that focus on modelling solutions on the conceptual level, and may constitute “building blocks” for your ontology, there are a variety of ways that these can be reused and applied. At one end of the scale, ODPs can be used similarly to design patterns in architecture, or how patterns many times are used in software engineering, i.e., as mere inspiration and a conceptual framework to keep in mind when designing your own solution. An example of this way of applying a pattern would be to read about its basic idea in a book, or an online catalogue, incorporate this idea into your own knowledge, and then proceed to design your artefact according to your own interpretation of that pattern, with any modifications you see fit. This way of reusing patterns is sometimes denoted *reuse by analogy*.

At the other extreme, some ODPs (in particular content ODPs) can be directly reusable through their OWL building blocks. This means that there is a small ontology available that represents the ODP, which you can directly import and use in your own ontology. This is similar to reusing an existing ontology, with the main difference that the ODP is usually small, has clear documentation of its capabilities, consequences, and so on, it is indeed designed for reuse, and ideally makes a minimal ontological commitment outside of its core purpose. In this case there is no way for the ontology engineer to make modifications to the ODP OWL-file, and you are affected by any changes that the ODP authors might make at a later date if you are resolving the ODP URI on the Web. Including the possibility of the ODP disappearing, i.e., that you at some point end up with a broken link in your ontology. However, the fact that ODPs are designed and published with reuse as their main purpose to some extent mitigates this risk, i.e., they change less frequently than other ontologies and are often hosted in ODP repositories, which reduces the risk of broken links.

Many people also follow some middle path between these extremes, potentially reusing the OWL building blocks of a few well-known and stable ODPs directly, but then creating their own “ODP catalogue” for their project for the rest of their needs, or even model the rest of the ontology in a more monolithic way. There are also ODP-based ontologies that do not import any ODP OWL

building blocks, but align to them through axioms of the ontology, e.g., expressing equivalences between locally defined classes and ODP classes. In the latter case it is up to the user of the ontology to decide how much of the ODPs to actually take into account, formally.

In [7] Gangemi and Presutti noted five types of operations that could be performed on ODPs when reusing them; *import*, *specialisation*, *generalisation*, *composition*, and *expansion*. *Import* denotes the reuse of the ODP as a component in the ontology being built, usually making use of the `owl:import` functionality. *Specialisation* can either take place in the mind of the ontology engineer, i.e., creating your own version of the ODP that specialises concepts and relations for your domain conceptually, but without actually importing any ODP “building block”. On the other hand, if the ODP is actually imported, specialisation can be done by creating subclasses and subproperties of the ODP classes and properties, and specialising its additional axioms using the new classes and properties. *Generalisation* is a less common operation, but may occur when creating or extracting ODPs from an existing ontology, hence, generalising a set of classes and properties to be reusable also in other contexts. *Composition* on the other hand is frequently used when applying ODPs. Because, unless you are creating a very small ontology, one single ODP is rarely enough to solve all the requirements of your intended ontology, and since ODPs focus on small pieces of the overall solution, you will inevitably have to compose, i.e., combine, several ODPs or ODP specialisations in order to solve the overall problem. *Expansion* is related to a similar task, i.e., to cover those parts of your requirements that are in fact not solvable by any ODP, or to extend the ontological commitment of an ODP while specialising it, in order to correctly capture the domain knowledge you are modelling.

Previous studies have also shown that there are various different ways of carrying out the above operations. One study focused on the specialisation task [10], and showed that there are two main strategies for specialising ODPs. In the property-oriented strategy both classes and properties are specialised, i.e., both subclasses and subproperties are created and usually domains and ranges of the subproperties are set to those new subclasses. One could view this as a more complete form of specialisation, since almost everything (except potentially some general axioms in the ODP) is specialised. Hence, this also opens up the possibility of simply removing the import of the ODP module at a later stage, while still retaining much of the semantics of the model. The other common strategy is the class-oriented one, where only classes are specialised, i.e., subclassed, and then restrictions are set on those classes in order to express their relation to the properties already defined in the ODP itself. Here, no new properties are defined, i.e., no subproperties of the ODP properties, and hence, the ontology built heavily relies on the imported ODP. Also hybrid variants are of course possible. The main advantage observed of the property-oriented strategy is that it makes the module created more self-contained, and more independent of the ODP, and axioms can be added to describe the new properties more in detail. The main advantage of the class-oriented strategy is the data interoperability that it creates, based on that the original properties are the ones that will be used in data expressed according to the ontology. However, there is a downside in terms of rea-

soning complexity, since the class restrictions commonly used with this strategy will usually increase reasoning complexity.

2.2. The eXtreme Design Methodology

The eXtreme Design (XD) methodology [15, 16] was initially proposed as one scenario in the context of the overall NeOn methodology [15]. Nevertheless, XD is sufficiently self-contained to be considered an ontology engineering methodology by itself, in the sense that it incorporates both project initiation steps, requirements analysis, development, testing and release of an ontology. Here the original version of the methodology is described, with some additions since its original publication, while in the following section some possible adaptations to the methodology for use in specific contexts are discussed.

2.2.1. Background and Underlying Principles

XD was originally proposed as a reaction to the focus on waterfall-like methodologies in ontology engineering, intending to introduce a new and more agile way of thinking about ontology engineering. It was initially inspired by methodologies from software engineering, such as the eXtreme Programming (XP) [17] and experience factory [2] approaches. However, despite being inspired by these, there are some main differences between software and ontology engineering that do not allow for a direct transfer of methodologies between the fields. One such difference being the fact that ontologies are always “white box”, i.e., a developer or tester needs to be aware of and concerned with the internal workings of a module, while in software engineering modules can at some point be considered “black box”, and the engineer needs mainly to be concerned with an interface exposing some functionality. Another difference is also the focus on conceptualisation in the case of ontology engineering, which may introduce a number of non-functional requirements for ontologies, e.g., in terms of domain coverage and accuracy, including naming of concepts, for instance. Naming does not affect the technical ability of an ontology to answer queries or perform reasoning, but it is highly relevant for humans to understand and make sense of both the ontology itself and the output of any task it is used for.

Nevertheless, XD has been inspired by agile software engineering methodologies, and applies a set of principles similar to those of, for instance, XP. As discussed in a later section, how closely such principles are followed certainly may depend on the context where the methodology is applied.

First of all, XD needs high involvement of a “customer”, i.e., someone who is going to use the ontology later on, either directly or indirectly. It should be noted that such a “customer” may not necessarily be the end user or person paying for the potential system to use the ontology (customer in an economical sense). In fact, since ontologies are rarely used in isolation, but rather as a part of some application or software system, the appropriate notion of “customer” may very well cover also the software developers who’s components are to exploit queries and inferences made by the ontology. Nevertheless, domain knowledge is

of course of essence when designing an ontology, hence, it is at least as important to also involve domain experts as customers in the development process. The customers are involved in requirements engineering, they have to confirm the detailed requirements developed, as well as verify that the end result corresponds to those requirements, e.g., both in terms of functionality as well as domain coverage, appropriateness of terminology, and other non-functional requirements.

Second, XD is highly requirements-driven, i.e., the ontology should contain exactly what is needed and nothing else. In earlier publications [15] this was denoted “task-focused”, meaning that ontologies built using XD are always focused on a certain set of tasks, rather than only being a representation of some knowledge domain, without any intended task for the formalisation of that knowledge. Note however, that tasks may be quite general, such as providing a schema for a set of data to be published on the web. Originally this was another reaction to some early ontology engineering methodologies that lacked such a requirement focus, and thereby allowed ontology engineering projects to go on for years and years, without ever knowing when the ontology would be “finished”. In order for requirements to be realistic, they should also be based on “stories” that come directly from the customers, i.e., rather than invented by the ontology engineers themselves, which is a common mistake in ontology engineering as well as in software engineering.

Third, the XD methodology is iterative, and incrementally builds the end result. This means that the methodology will produce a tangible result early on, while then extending that result in each iteration. When XD was first proposed, this was also a novelty in the ontology engineering community, where many previous methodologies and ontology projects had been focusing on first achieving a complete coverage of requirements, then a complete domain coverage, e.g., in terms of domain terminology, and only thereafter starting to actually formulate some ontological axioms. XD has a short time to first release, hence, it is particularly appropriate as a methodology for “rapid prototyping” of ontologies. When used not only as a prototyping methodology, this of course has the consequence of instead increasing the amount of work that has to be spent on resolving conflicts and refactoring the ontology in later iterations, due to both changing requirements and the design choices made in the newer modules being built. However, as will be noted later in this chapter, using ODPs is actually one way of partly mitigating the risk of having too diverse modules in the end.

Nevertheless, the ontologies created using XD will be inherently modular, since they are created piece by piece, and integrating the pieces one by one. Modularity can be an advantage in terms of easier reuse of parts of the ontology, and opening up the possibility to only use parts of the ontology to perform some “local” inferencing and querying. However, technically the structure of the ontology might become quite complex. In particular if each module is given its own namespace - which would be ideal if only considering the (re)use aspects mentioned. This may not be ideal from a human communication perspective, since an ontology could consist of tens or even hundreds of modules, it will become quite difficult to keep track of what namespace contains what module(s). In this case it may be more intuitive to simply use the “module” notion as a conceptual one, rather than technically separating them into separate ontologies.

Next, XD is test-focused, i.e., testing against requirements is a central part of the development. Whereas many agile software engineering methodologies are really test-driven, i.e., all tests are created even before the software is designed and built, this is not really possible when it comes to ontology engineering. The main reason is that since, as mentioned earlier, ontologies are “white box”, writing a test requires knowledge of the internal structure of the ontology, hence, writing a test then implies assuming a certain internal structure, thus designing the ontology itself. For this reason, only generic tests, at a structural level, can be created before the ontology is designed, as proposed in [14], while selecting which tests are applicable and how to best test the actual domain conceptualisation has to be done later. However, XD still has a clear test-focus, in the sense that all (functional) requirements written should be testable, so that the fulfilment of those requirements indicates the completion of a module or the ontology as whole. Of course, then XD suffers from the same difficulty as software engineering when it comes to testing non-functional requirements. In the ontology case a non-functional requirement could be the appropriate use of domain terminology in naming elements of the ontology. Although this is hard to test, and there may not even be a clear answer since terminology is not always agreed even among a set of domain experts, it is important to agree on a procedure to confirm also that such requirements have been met.

Another of the core principles of XD is reuse, and in particular reuse through ODPs. Ontology reuse in general is difficult, due to both overcommitment of the reused ontologies, and the sheer size of ontologies to understand and reuse, as discussed previously. Reusing ODP building blocks, or even ODPs just as ideas of solutions, is much more flexible but still lets the ontology engineer benefit from previous solutions. In addition, XD encourages also the development of ODPs, both specific and generic, which could be the starting point of a “component library” of reusable ontology modules, for instance within an organisation. Such a library can then constitute a “common language” for ontology design in that organisation, c.f. the notion of a pattern language [1]. However, this could also be useful even if applied only within a single ontology development project, in order to reduce the need for refactoring and alignment in the module integration phase.

In addition, the divide-and-conquer paradigm that is inherent to XD leads to a natural modularisation of both the problem, and its solution, which facilitates distributed ontology development, and assists in scoping the modelling issues that are addressed within a single iteration. However, to handle this incremental and potentially distributed ontology development, integration and refactoring become essential tasks. To some extent one could view XD as pushing some of the hard decisions of ontology engineering towards the end of development, instead of solving them before starting to model, i.e., conflicts and inconsistencies are only dealt with after they become a concrete problem in the integration of new modules, rather than beforehand. Taken to its extreme, developers of a certain module are not to take into account any requirements outside their small story and their set of requirements. However, in reality, and in particular when using a shared catalogue of ODPs, there may still be some communication around suitable design choices among development teams, which in turn may reduce the

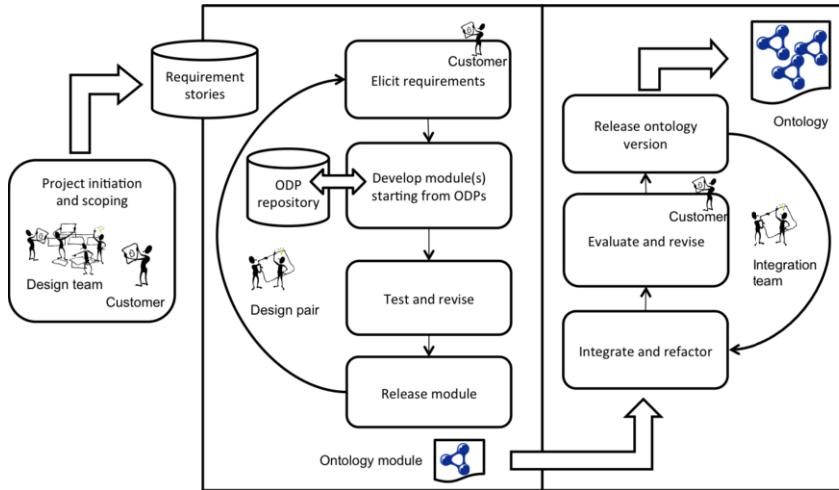


Figure 2.1. Overview of the XD workflow.

need for later refactoring.

The latter puts some focus on the collaboration model of XD, which promotes pair design, comparable to pair programming in XP, but tries to limit the interaction between design pairs during module development, in order to keep their focus and limit them to the scope set by their particular requirements. Nevertheless, since integration and refactoring is such a crucial part of the methodology, and this is not necessarily performed by the same set of people, documentation and proper communication of motivations of design solutions is essential. XD is suitable for distributed development, e.g., by geographically distributed teams that collaborate online, however, it is important to set up appropriate communication channels for sharing solutions, ODPs, and discussing refactoring issues.

In the following sections, the steps of the methodology are described in detail. Roughly the methodology can be divided into three parts; (1) a project initiation and scoping step, which is only performed once, at the start of the project (although may have to be revisited if conditions change), (2) a development loop that iteratively produces new modules, and (3) an integration loop that adds the increments to the overall solution. An overview of the methodology workflow can be seen in Fig. 2.1. In the following three subsections, first the project initiation and scoping steps are discussed, and thereafter details of the development and integration loops.

2.2.2. Project Initiation and Scoping

As with any development project, some general agreements have to be reached, if not given by the project specification itself. This can be things like staffing and distribution of roles among project participants, setup of an appropriate technical environment and decisions on representation languages and frameworks, agree-

ment on the exact procedures to be followed within the project, including release plan and integration strategies, project scope and priorities, reuse opportunities, as well as a timeline with deadlines and milestones. At a high level an ontology engineering project does not differ much from any other development project. Nevertheless, some of the mentioned things to consider has a specific meaning in the context of ontology engineering and XD, which are worth mentioning.

One thing that sets ontology engineering apart from, for instance, many software engineering projects, is the need for a deeper understanding of the target domain, even among the developers and not only at the management level. This is due to that the knowledge to be modelled in the ontology usually needs to be understood to a larger degree by the ontology engineers in order to, for instance, use correct terminology in the ontology, and make appropriate design choices that correctly solves the set of requirements. Again, this boils down to the fact that an ontology is “white box”, hence the users (whether software developers of the surrounding system, or end users of the system) may have to in turn understand and approve of the inner workings of what is constructed. The consequence of this is that there is a greater need for developing a shared understanding of the domain, its terminology, the intended tasks of the ontology and so on, in an ontology engineering project than in, for instance, many software projects. Therefore this should be taken seriously already from the project start, and this activity is also to be revisited throughout the project in order to clarify any potential misconceptions. Commonly, a shared tool is set up for the purpose of documenting and discussing both domain knowledge and requirements, e.g., the user stories, so that domain experts can closely follow and also participate in any discussions, even if distributed in terms of time and location. Such a tool could constitute a simple wiki, or a more advanced project management system coupled with requirements analysis tooling.

Additionally, scoping is very important for ontologies, but also very hard to clearly define in terms of the knowledge domain to be modelled. Here the task-focus of XD can be very helpful, allowing to focus on the generic tasks that the ontology should support, rather than the domain coverage in terms of concepts, attributes and terminology. This could for example be specified through its use by other system components, its roles in a system to be built, while more or less considering the ontology and its associated querying and inference software as a black box from the system point of view. Of course, there may also be ontology engineering projects with less clear scope, and where it is not yet clear what the ontology might be used for, but then it may be hard to use XD as the main methodology without some adaptation.

Further, before starting the actual development, one needs to agree on the starting point of the project, e.g. in terms of any existing resources to take into account, or even reuse, what ODP catalogue(s) to use, and how to manage the shared set of ODPs that will emerge during the XD process. It is rarely the case that ontologies are constructed completely from scratch. Usually there are at least some legacy systems or terminologies to take into account, e.g., anything from implicitly shared vocabularies within an organisation, via legacy databases, to standards and already existing ontologies. How each such resource is to be managed has to be determined at the start of the project. Procedures that need

to be made explicit could include to what extent ontology modules and tests are to be documented, what naming conventions are to be used, where files are stored and how versioning is to be managed, how the integration process is to be managed, quality assurance processes, and when and what to include in an ontology release.

At this stage, also other non-functional requirements of the ontology may be listed. These can be overarching requirements, such as how well the domain terminology needs to be represented in the naming of concepts and properties, how much documentation the ontology should contain and how that documentation should be written to serve later purposes, e.g., to display information in a user interface, any trade-offs between domain coverage and functionality that should be observed, requirements on the overall architecture of the ontology, the OWL-profile to be used etc.

Finally, before starting the development loop, some user stories need to be developed. User stories will later lead to the development of the functional requirements of each ontology module. Stories can be formulated in different ways, e.g., as examples of data for which the ontology is to act as a schema, or describing some functionality that is to be realised based on the ontology (for the latter c.f. the “original story” in Table 2.1). The important thing is to keep them short and focused, i.e., on one concrete part of the domain knowledge, one specific task. If stories are too big they will have to be broken up into smaller stories before starting the development loop, in order to avoid the situation where one development pair would more or less develop the whole ontology. A typical story might contain anything from 1-2 sentences up to about two brief paragraphs of text. Additionally, stories need to be quite specific in order not to allow for too much interpretation by the ontology engineers. Stories can be written in collaboration between “customers” and ontology engineers, but should be driven entirely by the needs of the customers.

An example story from a research project where XD was applied² can be seen in Table 2.1. The context of the ontology is an “intelligent bathroom” environment that should proactively serve the user with context-dependent information related to current user needs and preferences. Since the original story contained quite a few aspects and tasks of the resulting ontology, it was in this case broken down into 10 smaller stories (only a few examples are shown in the figure for space reasons) that were made more specific, identifying exactly what was the task of the system in each step. These 10 stories were then used as the basis for the development of a set of ontology modules that would together support the functionality described in the original story.

All stories should then be organised in terms of priority, and possible dependencies between them are identified and made explicit. It is therefore suitable for each story to be described by means of a small card, like the one depicted in Table 2.2, which includes the unique title of the story, a list of other stories that it depends on, a description in natural language, i.e., the story itself, and a priority value. The example story in Table 2.2 comes from a course on ontologies in the legal domain³. Typically these stories can be collected using a form in a

²<http://www.iks-project.eu/>

³http://ontologydesignpatterns.org/wiki/Training:Legal_Ontology_Design

Table 2.1. Example of the breakdown of a story that was originally too large, in terms of covering too many aspects and tasks of the ontology to be built.

Original story:	It's Thursday morning. I get site-specific weather information when I am brushing my teeth in the bathroom. Based on weather information and my calendar, free-time event suggestions are given (e.g. "Today, 8 p.m. - Sneak Preview at CinemaOne). Do you want to order tickets?
Substory 1:	I am alone in the bathroom. I am standing facing the mirror with the electric toothbrush in my hand, hence, the system recognises that I am brushing my teeth.
Substory 2:	I was brushing my teeth in the bathroom on Thursday morning. Morning is between 6 and 10am according to the current user.
Substory 3:	I am living in Berlin and I like to get the local weather displayed in the morning as soon as I am brushing my teeth.
...	...
Substory 10:	The system asks me if I want to order tickets to a proposed event if there are tickets available. I am presented with a set of ticket options, and get to select the time of the show, the seat I want at the movie theatre. Finally, I pay by credit card.

Table 2.2. A proposed template for user stories, filled with an example story from the legal domain.

Title	Founding of legal entities
Dependencies	...
Priority	High
Story	FIAT (Fabbrica Italiana Automobili Torino) was founded in Turin in 1899 by Bricherasio, Goria, Biscaretti, Ferrero, Ceriana-Mayneri, Racca, Scarfotti, Damerino, Agnelli and the Bank di Sconto e Sete. The deed of constitution of FIAT, showing the Savoy coat of arms and the original company name on its cover, was drafted by the public notary Ernesto Torretta, signed by all the founders, and registered on the 11th July 1899.

wiki setting, or a more elaborate requirements management system may be used. The two examples given here in Table 2.1 and 2.2 both phrase the story as a concrete example, i.e., containing elements that would be part of the knowledge base rather than the ontology. Although stories can very well be expressed in more general terms, it may often be easier for domain experts to provide concrete examples of data and how the system should react to that data, rather than providing descriptions of generic situations. Of course this comes with the risk of missing something generic that does not happen to be covered by the concrete example.

Since XD is agile and iterative, it is not necessary to develop all user stories beforehand, but an initial “backlog” is to be accumulated before starting the development process. This is to ensure that an appropriate prioritisation can be made between the initial set of stories. Simply picking up the first one being written may result in development starting with a very low-priority story that otherwise may not even have been included in the ontology at all. At this stage,

it is also important to constantly update the agreement with the “customers”, unless otherwise specified by formal constraints, such as a contract. As the set of stories is allowed to emerge and evolve over time, it is important to also update the agreement of what is actually going to be implemented. For this purpose, there are two main points of agreement between the “customers” and the project management; one is the agreement on what stories are to be prioritised, and in the end even what proposed stories to implement at all, and the second one is at the level of agreeing on the detailed requirements of each story (see further in the next section).

2.2.3. Module Development Loop

Once some user stories have been collected and prioritised, the concrete development of the ontology can begin. As mentioned previously, this is done incrementally, one module at a time. Ideally, each story will correspond to one (or a small set of) ontology modules, however, the situation may also occur that some stories are considered too overlapping, so that their solutions have to be merged.

Nevertheless, the development of the stories can be done in parallel by as many design pairs, i.e., pairs consisting of two ontology engineers, as the project has access to. Pairs may of course communicate with each other, but ideally such communication is reduced to a minimum, and the focus of each pair should mainly be on their own story only, regardless of the requirements developed from other stories. This is important in order to avoid pairs getting stuck on issues that should actually be resolved later on, instead of developing what they feel is the best solution for their small sub-problem.

The module development loop is represented in Figure 2.1 by the left rectangle, where a design pair loops through the activities of requirements elicitation, module development, testing, and module release, for one story at a time. The first step is for the design pair to pick up a new story to work on. How this selection is done may vary, but priority of the story, dependencies on previous stories treated by the pair, and the skills and competencies of the pair, may impact the selection.

Once selected, the pair should perform requirements elicitation from their story. Of course, this process may need considerable involvement by the customers (who wrote the story) in order to interpret its intended meaning and ensure an appropriate coverage of this particular subset of the domain. If the story was formulated as a concrete example, e.g., as exemplified in the previous section, it is now time to generalise it. One way to perform such generalisation is to first reformulate the story sentences into *instance-free* sentences, i.e., to for each mention of a named individual or example attribute value, replace that with a term representing the expected type of this individual or a potential attribute that could hold that value. Table 2.3 shows an example of such a transformation, based on the example story from the legal domain presented earlier.

Once the story text has been sufficiently generalised, a set of requirements should be elicited from it. These requirements commonly fall into the following three categories:

- Competency Questions (CQs)

Table 2.3. Example of how a set of phrases from a user story may be generalised.

Original phrase	Example of instance-free phrase
FIAT (Fabbrica Italiana Automobili Torino) was founded in Turin in 1899 ...	Legal persons are founded in a certain location at a certain time ...
... by Bricherasio, Goria, Biscaretti, Ferrero, Ceriana-Mayneri, Racca, Scarfiotti, Damerino, Agnelli and the Bank di Sconto e Sete.	... by a set of legal persons.
The deed of constitution of FIAT, ...	Legal entities are created by a deed of constitution ...
... showing the Savoy coat of arms and the original company name on its cover, which consists of text and pictures, ...
... was drafted by the public notary Ernesto Torretta, and is drafted by a public notary, ...
... signed by all the founders, and registered on the 11th July 1899.	it is signed by the founder persons and a certain point in time.

- Contextual Statements (CS)
- Reasoning Requirements (RR)

CQs are probably the most well-known category of ontological requirements, which was recognised already at the very beginning of the knowledge engineering tradition [8]. CQs express typical tasks of the ontology, i.e., typical queries it should be able to answer, and are usually expressed as natural language sentences, e.g., questions. However, through applying XD it has been noted repeatedly that CQs on their own do not always suffice in order to clearly specify what is required from the ontology, especially in the following two respects:

- Are there any constraints that should be enforced over this knowledge, or any common-sense notions that are to be introduced to complement the knowledge needed to answer the CQ? - Answers are CS
- Is all the information needed to answer the CQ going to be entered explicitly into the knowledge base, or is there some inferences required either in order to derive the answer to the CQ or that should be derived as a consequence of the response? - Answers are RR

Note that both of these questions refer to the CQ, hence the CQs are the requirements that set the scope of the module to be built and drive the need for additional requirements. However, CS and RR are sometimes needed in order to precisely specify the additional axioms of the entities mentioned in the CQs that are needed in order for the ontology to perform a certain task. Considering a CS, the task may be consistency checking, or identity resolution - in addition to answering the CQ. While considering an RR, the task may for example be classification of instances, in order to then be able to answer the CQ based on the inferred knowledge. In Table 2.4 some example requirements are presented based on two of the example stories seen previously.

Before leaving the requirements elicitation activity, the set of requirements should be “signed off” by the customer, i.e., an agreement should be reached that these requirements are necessary and sufficient for considering the story as

Table 2.4. Example breakdown of stories into detailed requirements.

Story	I am alone in the bathroom. I am standing facing the mirror with the electric toothbrush in my hand, hence, the system recognises that I am brushing my teeth.
CQ	Associated CS or RR
Who is where in this indoor location?	-
What sensor data is known about this user's context?	-
What is the user doing now?	RR: Activity inferred based on the available context information, from a fixed set of activities.
Story	FIAT (Fabbrica Italiana Automobili Torino) was founded in Turin in 1899 by Bricherasio, Goria, Biscaretti, Ferrero, Ceriana-Mayneri, Racca, Scarfiotti, Damerino, Agnelli and the Bank di Sconto e Sete. The deed of constitution of FIAT, showing the Savoy coat of arms and the original company name on its cover, was drafted by the public notary Ernesto Torretta, signed by all the founders, and registered on the 11th July 1899.
CQ	Associated CS or RR
Who founded a legal person?	CS: The “who” must also be a legal person.
When was a legal person founded?	CS: Each legal person was founded at exactly one point in time.
Where was a legal person founded?	CS: Each legal person was founded at exactly one location.
Who signed the deed of constitution of a certain legal person?	CS: Each deed of constitution is signed by at least one founding person.
What is contained in a deed of constitution?	CS: Content can be of types text or images.
Who prepared (drafted) a deed of constitution?	CS: Each deed of constitution was drafted by one or more persons.
When and where was a deed signed and registered?	RR: A deed is valid when it has been both signed and registered.

covered by a solution. This means that when these requirements are confirmed through appropriate testing the solution module can be considered complete.

The following development step constitutes the actual modelling, i.e., creating a solution covering all the requirements of this story. Depending on the size of the set of requirements of a story, and how disparate they are, it may at this stage be a good idea to select only one or a few of the CQs to treat initially, creating an incremental building process for each module as well. Once a small set of coherent CQs have been selected, the first task is to look for any existing ODPs that may match the requirements at hand. This matching task is supported by the fact that most Content ODPs are annotated with a set of CQs in themselves. However, due to the gap in abstraction, i.e., ODPs being usually quite abstract solutions while the requirements at hand are usually much more concrete and domain-specific, providing good tool support for this matching process is difficult and currently it is therefore mainly a manual task. ODP repositories, such as the ODP portal⁴ can be browsed and searched for potentially relevant ODPs, and some tools also provide search functionalities to perform keyword search over ODPs and their annotations [12]. Nevertheless, usually the ODPs found then have to be manually assessed and compared to find the best match for a particular situation. Commonly certain abstract notions, such as events, participation, states and so on, can be represented in several alternative ways, whereas several ODPs exist for such notions. In such cases the ODPs need to be carefully examined, consequences considered, and finally the solution (if any) that best matches the requirements at hand can be selected.

Once an ODP has been selected, the next step is usually to specialise that ODP for the domain problem at hand. In rare cases an ODP may be suitable to use as-is, but this is not commonly the case. As described earlier, there can be several ways to perform the specialisation, e.g., both by reusing the ODP simply as an abstract template and source of inspiration, or by reusing the actual building block, and if reusing the actual building block there are also various strategies for specialisation. Depending on the tool used for modelling, there may be more or less support for this task in the tool. Usually ontology engineering environments support `owl:import` axioms, which can be used to import ODPs as components in your ontology module. However, some tools do not at the time of writing support imports, such as the WebProtégé tool, however, for this tool in particular there is a specific XD plugin being built to partly remedy this situation [12]. In addition to specialisation, covering all the selected requirements usually requires either some composition of several specialised ODPs, or an extension to the ODPs . If a considerable portion of the solution is not supported by any ODP, one may also consider to extract and potentially generalise that solution, and propose it as a new ODP . Taking the time to do so, and sharing it at least with the fellow ontology engineers of the same project may potentially give rise to less problems in the integration of modules later on, since other design pairs may then reuse the same generic solution for their specific modelling problems.

An example ODP specialisation, covering the first CQ of the story from the legal domain in Table 2.4, can be seen in Figure 2.2. The `ParticipantRole` ODP⁵

⁴<http://ontologydesignpatterns.org/>

⁵<http://ontologydesignpatterns.org/wiki/Submissions:ParticipantRole>

has been used, and specialised by adding classes specifically representing legal roles, legal persons, and legal events, as well as the central class (**LegalParticipantRole**) representing the “n-ary relation” that connects a certain legal person to a certain legal role in a certain event. Assuming that “founding” is a legal event for which this specialised model will now be used, then the CS follows from the range restriction set on the **legalPersonParticipating** property, i.e., all participating objects pointed to by this property will be inferred to be legal persons. Whether this is exactly the intended behaviour for the CS may have to be confirmed with the customer, but let us assume it is for the sake of this example.

Once a module has been developed, it should be tested against all its requirements, c.f. [5]. CQs are usually tested through SPARQL queries, i.e., creating a test module, importing the module that has been built and adding test data, then writing and running SPARQL queries corresponding to the CQs over the module. Although this seems quite straight-forward, deciding if the model appropriately solves the CQs may require a bit of deliberation. In some cases it may be possible to write the SPARQL query, but the query turns out to be overly complex, or contains implicit assumptions that could instead be made explicit in the model. Hence, both the query itself and the retrieved results should be analysed in order to decide whether the module has passed the test or not. A test is considered successful if an error is found, i.e., data is missing or different from the expected results. This helps finding mistakes in the model or parts of the specification that were overlooked during the design. In Figure 2.3 the module with some added test data is illustrated, and Figure 2.4 contains a SPARQL query that could be used as a test case for testing the CQ with the test data in the figure. If the original story was written in an exemplary form, then realistic values and test instances may be taken from it.

Reasoning requirements are often as straight-forward to test as CQs, i.e., the ontology engineer simply has to enter some test instances and run an appropriate inference engine over the ontology with the added test data and check the inferred statements against an expected set of statements. The important thing to look for, however, is not only the expected inferences but especially the unexpected ones. It is the unexpected inferences that indicate a problem in the design.

Finally, the contextual statements are the most difficult to check. Preferably they should be accompanied by some explanation of the purpose for which they should be expressed in the ontology, whereas that purpose could be tested. Some CS are also expected to generate inconsistencies under certain conditions, or added inferences, such as inferring **owl:sameAs** axioms holding between individuals. In the latter case they can be tested in a similar manner as the RRs.

Whatever test cases and test data is developed, it is important to document and save all of this for later use in the integration phase. Integration of the module into the overall ontology may affect the result of the tests, which should be rerun after integration, c.f. regression testing of software, where previous tests are run again to ensure that modifications or newly integrated software has not affected the already integrated modules. Additionally, after any refactoring needed in the integration phase, the test cases (potentially modified if the module has been refactored) should also be ran again, with the same test data, in order to check that the same results are still achieved.

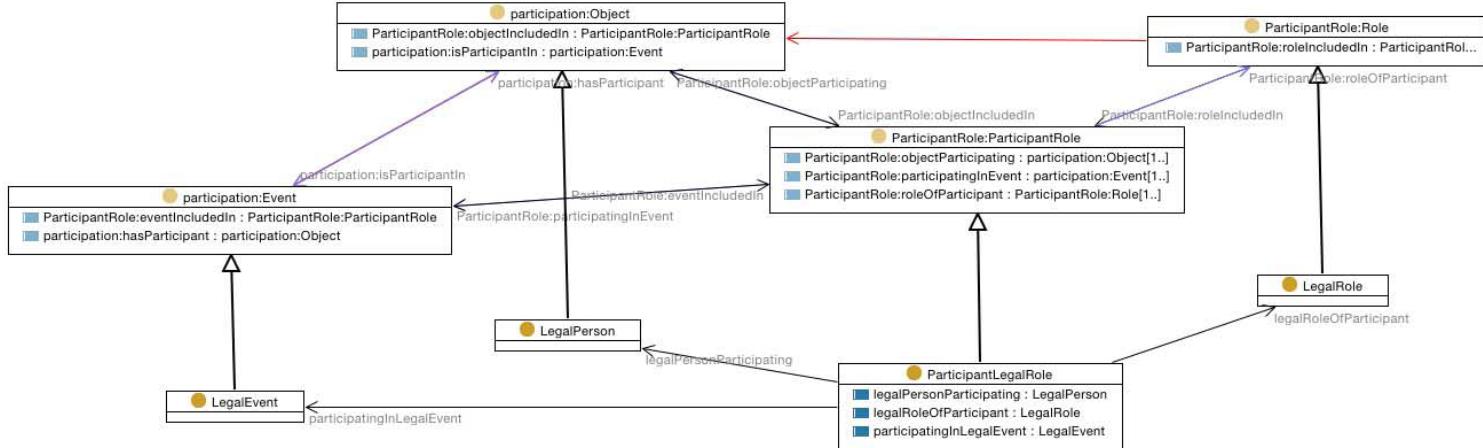


Figure 2.2. An illustration of a specialisation of the ParticipantRole ODP for the legal domain, concerning participation in legal events, such as the founding of a legal entity. Illustration is using the UML-inspired OWL notation of TopBraid Composer. The four classes at the top of the figure constitute classes from the ODP, while the bottom four constitute the specialisation, including the three specialised properties that can be seen connecting these classes.

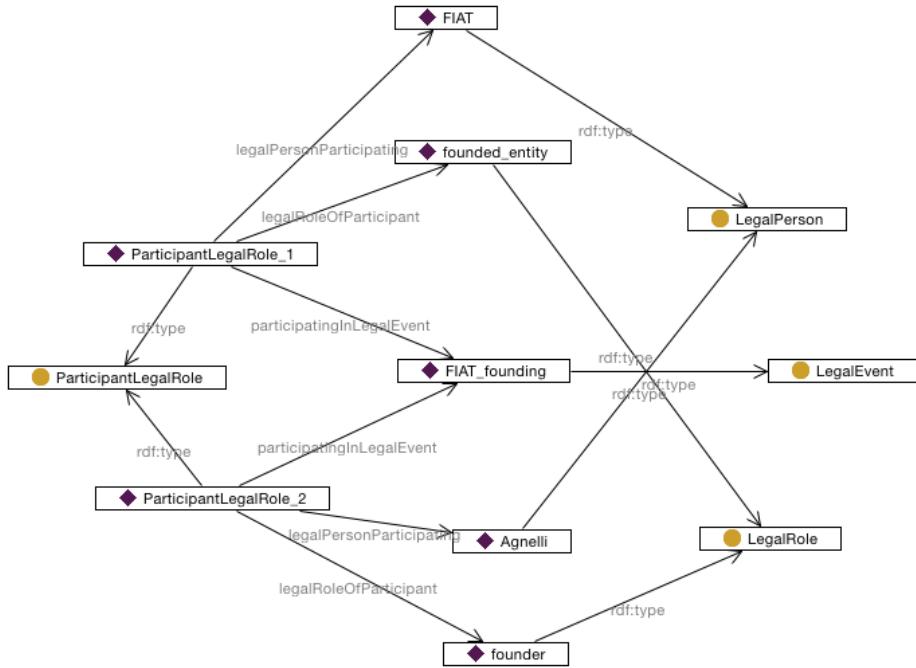


Figure 2.3. An RDF graph representing some test data expressed using the example ODP specialisation in Figure 2.2. The boxes with diamonds represent individuals, while the ones with circles represent classes, i.e., the types of the individuals, in the graphical notation of TopBraid Composer. The data graph could be read as “FIAT (a legal person) participated as the founded entity (legal role) in the FIAT founding event” and “Agnelli (a legal person) participated as a founder (legal role) in the FIAT founding event”

```

SELECT ?person ?founded
WHERE {
?participation1 p:legalPersonParticipating ?person .
?participation1 p:legalRoleOfParticipant :founder .
?participation1 p:participatingInLegalEvent ?event .
?participation2 p:legalPersonParticipating ?founded .
?participation2 p:legalRoleOfParticipant :founded_entity .
?participation2 p:participatingInLegalEvent ?event
}

```

Figure 2.4. An example SPARQL query that could constitute the test case for the first CQ of the legal story in Table 2.4, assuming the model as in Figure 2.2 (represented by prefix p:) and the example data as in Figure 2.3. Given the example data, the query should return the instance pair Agnelli and FIAT, i.e., representing the fact that Agnelli was a co-founder of FIAT.

Once all test cases for all the requirements have been executed, and no further issues have been discovered, the module can be released. The module should of course be appropriately documented, according to the project-specific guidelines set up at project initiation, e.g., by using annotation properties inside the ontology file.

2.2.4. Integration Loop

The integration loop is represented by the righthand side of Figure 2.1. Integration may be performed by a dedicated integration team, or by the design pairs themselves. The advantage of having a dedicated team taking over the released modules is that this team can specialise in integration and refactoring, and also maintain an overview of the overall ontology so far. It also forces the design pairs to document their modules in such a way that they can be immediately understood by others, which may not be the case if they are responsible for integrating their module themselves.

If the integration is performed between a module and a large ontology it may be hard to find all the suitable points of alignment manually. For this task there are numerous ontology alignment systems available, which to some extent have been tested for use within XD [6]. However, so far XD tooling, such as the WebProtégé plugin [12] that will be discussed later, only apply a naive matching approach for proposing alignments, i.e., string matching of names and labels to find potentially equivalent classes. This means that a large part of the alignment task may have to be done manually by the integration team, relying on their expertise.

Additionally, they are responsible for making decisions on refactoring, and implementing such refactoring. For example, if an overlap between two modules are found, even if the design solutions are perfectly compatible one has to make a decision on what strategy to use in order to align the modules. One strategy could be to keep the classes as they are in each module and add equivalent class axioms between them. This has the advantage of keeping the modules self-contained, making it easier to use modules independently for certain reasoning tasks, and facilitates reuse of single modules in later ontology engineering projects. However, the resulting ontology may not be very intuitive to a human, due to the presence of duplicate classes (potentially with different names), which may increase the risk of misuse of the ontology, and which may additionally increase the complexity of querying for data without first materialising all inferences. The situation may also occur when something is missing, in order to connect modules, where the integration team then would have to perform some modelling in order to create the “glue” to fit the modules together.

What is even more difficult of course is the situation when modules apply completely incompatible designs, where one module really needs complete refactoring in order for both solutions to be incorporated in the overall ontology. There is no universal solution to this problem, each project needs to find the strategy that seems most suitable for their needs. However, any team applying XD needs to be aware of that this is one of the most crucial, and difficult, steps of the methodology, which will need quite a bit of time and effort, and potentially discussions

throughout the engineering team, and with the customer as well.

Fortunately, the use of ODPs is a crucial facilitator that makes this approach actually feasible in practice. With a limited ODP catalogue to start from, and potentially an additional emerging set of project-specific ODPs that are shared within the project, design solutions usually tend to converge rather than diverge as the project progresses. The ODPs constitute the common ground on which to build modules, and also constitute a common language for easier communication about solution alternatives. Once the ontology engineers become familiar with the ODP catalogue used, the ODP names are commonly used to signify design alternatives. A typical discussion among ontology engineers could be as follows:

- “I am thinking of using `AgentRole` for this module, I know you did another one involving roles recently, what ODP did you use?”
- “I needed roles to vary over situations so I decided to use `ParticipantRole` instead.”
- “Then I’ll use that as well, so our modules are compatible.”

Without actually talking about the details of the modelling choices they make, the two ontology engineers have agreed on a common design by means of referring to ODPs that they both know. This is the original idea of a pattern language as presented in architecture by C. Alexander [1].

As soon as a new module has been integrated into the overall ontology, it has to be evaluated . At this point, the integration team should reuse all the test cases for all the currently integrated modules and make sure they can be successfully completed also using the integrated ontology, i.e., perform regression testing. In addition, some new test cases should be created, in order to evaluate the effects of the integration, e.g., the added alignments. Finally, the ontology should be applied in its intended usage setting, e.g., within a software system, verifying the functionality that it is supposed to provide according to the stories covered so far. At this stage, it is important to involve the customers again, whether they are the developers of the surrounding software or end users, in order for them to agree with and “accept” the verification results. By involving the customers also any non-functional requirements can be verified, such as correct use of terminology, understandability of the model etc. Only once all such verifications, and potential revisions as a result of that, have been completed successfully a new ontology version is released.

The release process may involve some formal delivery of the ontology, e.g., by making it available online, and it may include to provide a new version IRI for the ontology (c.f. the versioning guidelines of W3C⁶). How often releases are made may also depend on the project setup. If modules are delivered and integrated very frequently, then it may be more reasonable to provide a new release of the ontology after integrating a set of new modules, rather than a single one.

⁶https://www.w3.org/TR/owl2-syntax/#Versioning_of_OWL_2_Ontologies

2.3. Tools and Support Infrastructure

At the time of writing, there is no ontology engineering environment that provides complete support for the whole XD process. However, there are various tools that provide support for certain parts of the process, which are described here. Additionally, some related infrastructure, e.g., in terms of pattern repositories that may also support the process, are also briefly introduced.

There are a number of ontology engineering environments available today, where the most commonly used probably are Protégé Desktop⁷, WebProtégé⁸, and TopBraid Composer⁹. Additionally, ontologies are also sometimes created through specifying them from an API, e.g., the OWL API¹⁰ or Jena¹¹, which then generates the OWL file, however, in this section the focus is on tools with graphical user interfaces, and which are not only intended for expert users. One additional ontology engineering environment worth mentioning is the NeOn toolkit¹², which was built by the NeOn project and for a few years after that still maintained by the NeOn foundation. However, as far as the authors are aware the NeOn toolkit is no longer being maintained and the latest version was released in 2011. Despite the fact that the NeOn toolkit actually has a plugin directly supporting XD, it will not be described here, both due to the uncertain status of the NeOn toolkit itself but mainly the fact that most of the functionality provided by that plugin has now been reimplemented in the XD plugin for WebProtégé.

First of all, it should be mentioned that most of the tasks of the XD methodology can be performed using either a general-purpose ontology engineering environment, such as Protégé or TopBraid Composer, with the addition of some support tools for specific purposes, such as requirements management, debugging, version management etc. This is indeed, in the authors' experience, how most projects have been set up when using XD. A typical setup could consist of the following:

- a **wiki** for project communication, documentation, requirements management, and collection of project specific ODPs,
- a **Git** or **SVN** repository for version control of the modules and the overall ontology,
- an **ontology engineering IDE** like Protégé or TopBraid Composer for the modelling of modules and integration of modules,
- a **reasoner** and **SPARQL** engine (potentially the ones shipped with TopBraid Composer) for testing,
- and the online **ODP repository** at ontologydesignpatterns.org as the main source of ODPs.

In order to further support some of the specific tasks of XD, the eXtreme Design for WebProtégé (XDP) extension¹³ has been developed [12]. XDP is a

⁷<http://protege.stanford.edu/products.php#desktop-protege>

⁸<http://protege.stanford.edu/products.php#web-protege>

⁹<http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/>

¹⁰<http://owlapi.sourceforge.net/>

¹¹<https://jena.apache.org/>

¹²<http://neon-toolkit.org/>

¹³Online demo: <http://wp.xd-protege.com>, video walkthrough:

reimplementation of many of the functionalities originally provided by the XD Tools plugin for the NeOn Toolkit, but also provides a set of novel new developments¹⁴, including a new search engine for finding suitable ODPs [11], some specific support for the different specialisation strategies (c.f. Section 2.1 and [10]), and an ontology alignment component for providing suggestions of points of integration after ODP specialisation [11].

Additionally, in [6] some existing ontology alignment and debugging tools are tested and analysed, which could further support the integration and testing tasks within XD. Further, the recent Protégé plugin for applying generic structural test cases for OWL ontologies [14], may also be useful for testing purposes. For refactoring of ontologies, the tool suite for pattern-based transformation of ontologies¹⁵ originating from the Patomat project may be used [18]. The Patomat tools allow for expressing transformation rules to transform an ontology, or an ontology module, from the design proposed by one ODP to a structure conforming to a different ODP. Since this requires some manual effort in terms of writing transformation rules it may not be suitable for a one-time effort, but for a large ontology engineering project where multiple modules need to be transformed, this may be a very useful tool to apply. It would also be possible to set up your own transformation services based on the OPPL language[13], or use the OPPL Protégé plugin¹⁶ for specifying transformations and templates directly.

Finally, in order to apply XD the project has to have access to appropriate ODPs. There are several ODP repositories available for ODPs of different types, but for Content ODPs the two most frequently used repositories are the ODP portal’s list of submitted Content ODPs¹⁷ and the Manchester ODP repository¹⁸. At the time of writing the ODP portal contains a list of 114 submitted Content ODPs. However, it should be noted that quality assurance of ODPs is not standardised, and the different portals may apply varying criteria for publishing an ODP. One quality assurance opportunity is to submit the ODP to the WOP workshop series¹⁹, where the ODP would undergo a peer review process. The ODP portal additionally provides various other ODP types, in addition to Content ODPs, as does the Manchester repository.

2.4. eXtreme Design Use Cases and Lessons Learned

In this section a brief description of some of the lessons learned from using ODPs, and the XD methodology for ontology engineering, are presented. First, a few projects and use cases where XD was applied are mentioned, both providing a brief history of some of the first large-scale applications of XD as well as providing an example of the variation of contexts and kinds of ontologies XD has been

<https://youtu.be/ZRH6vGXocqU>, code: <https://github.com/hammar/webprotege>

¹⁴Note that as at the time of writing WebProtégé does not support `owl:imports`, import and specialisation has been implemented through duplicating ODP entities in the target ontology.

¹⁵<http://owl.vse.cz:8080/patomat/>

¹⁶<https://sourceforge.net/projects/oppl2/files/>

¹⁷<http://ontologydesignpatterns.org/wiki/Submissions:ContentOPs>

¹⁸<http://www.gong.manchester.ac.uk/odp/html/>

¹⁹<http://ontologydesignpatterns.org/wiki/WOP:Main>

applied to build. Next, a number of experiments and user studies that have been carried out are summarised, which have analysed and pointed at the benefits of using ODPs and XD, and then a set of points of adaptation are described, where XD has sometimes been adapted to practical constraints with a successful result. Finally, some experiences in using ODPs and XD as tools for teaching and learning ontology engineering are presented.

2.4.1. Example Use Cases

XD has been frequently applied in various contexts over the past decade. In this section this usage is exemplified by briefly mentioning a few of the projects where XD has been applied in various forms. The intention is neither to provide a full listing of all projects where it has been applied, nor to describe each case in detail, but merely to exemplify the variety of ontologies and project contexts where the methodology has been used. Some of these projects were also used for studying aspects of XD more in-depth, as described in the next section.

XD was originally proposed in the context of the NeOn project, and ODPs and an initial version of XD was then applied to construct ontologies for one of the project use cases in the fishery domain. One of the project use case partners was the Food and Agriculture Organisation (FAO) of the United Nations, who were building a system for fish stock monitoring all over the world. The ontologies of this use case were constructed according to an early version of the XD methodology, still including the focus on requirements, modularisation and ODPs. The resulting network of fishery ontologies, i.e., ontology modules, was evaluated and published by the FAO²⁰, and act mainly as standardised vocabularies for information sharing and integration. This project also resulted in a set of “fishery ODPs” that are listed in the ODP portal²¹.

The XD methodology was also applied in a use case of the IKS project²², which was previously mentioned. The overall project focused on introducing semantic technologies into the domain of Content Management Systems (CMS), while the particular use case where XD and ODPs were applied focused on an ambient intelligence application, i.e., an “intelligent bathroom” environment that should proactively serve the user with context-dependent information related to current user needs and preferences. A physical bathroom environment was set up in a lab environment, where also the software using the ontologies was tested with actual users. In contrast to the NeOn use case, the ontologies created here were focused on reasoning tasks, in order to support the system “intelligence”, rather than merely acting as vocabularies for data. Hence, it was mainly in the context of this project that the different types of requirements (CQs, CS and RR) were elaborated, and different testing methods for them were introduced. A set of ontology modules, representing the ontology for supporting a few larger user stories that were tested in the bathroom environment are still available for download²³.

²⁰<http://aims.fao.org/network-fisheries-ontologies>

²¹<http://ontologydesignpatterns.org/wiki/Community:Fishery>

²²<http://www.iks-project.eu/>

²³<http://www.ontologydesignpatterns.org/iks/ami/2011/02/>

More recently, XD is being applied in a number of ongoing projects ranging from the creation of criminal profiling ontologies for criminal intelligence analysis²⁴, via ontologies for automated test case generation in software engineering (project OSTAG), to the development of Decision Making ODP prototypes in the context of a W3C incubator group²⁵, and various Linked Data projects. In several of these projects, XD was slightly adapted, e.g., in terms of personnel, roles, and project setup, while still maintaining the core ideas of requirement and ODP focus, as described further in Section 2.4.3.

2.4.2. Empirical Evaluations of ODP Usage and XD

In order to establish ODP usage and the XD methodology as a viable support to ontology engineers, a series of initial experiments and observational studies were conducted over the course of several years. The results of these studies have been reported in [3, 4, 9].

Conclusions include that ODP usage indeed improved several quality aspects of the resulting ontologies as expected, e.g., overall a lower error rate, higher understandability due to increased number of comments etc., although there are also a few pitfalls concerning misuse of ODPs, as noted in [9]. While it was not possible to quantitatively see any difference in the modelling time spent to solve a certain modelling problem, nevertheless, as noted in [9] the ontology engineers themselves feel that they solve the tasks faster. One potential explanation for this is that the time it takes to find, understand, and select an appropriate ODP without any specific tool support, and without prior experience with the ODPs in question, roughly corresponds to the time saved in the actual modelling phase when reusing the building block of that ODP. However, this is still an unconfirmed hypothesis.

Concerning XD, observations and questionnaires with ontology engineers using the methodology has led to the conclusion that most of them found the methodology very useful. An interesting note is that many subjects expressed that they were already working in a similar manner, e.g., in a divide-and-conquer process when addressing a modelling problem, but still their results improved considerably when XD was explicitly introduced. In this case a potential explanation could be the formalisation of the testing process, i.e., it could be the case that people think they test all their requirements properly, but in reality they may not actually be doing that if they are not following a structure approach for requirements management and testing. Yet again, this is an unconfirmed hypothesis.

2.4.3. Adaptations of the Methodology

So far the XD methodology has been described as it was originally proposed, only with the addition of some more details and a few clarifications compared to how it was described in [15, 16]. However, during the past 10 years a lot of experiences

²⁴Project VALCRI - <http://valcri.org/>

²⁵https://www.w3.org/2005/Incubator/decision/wiki/Draft_Final_Report

have been collected while using the methodology, some of which are described in section 2.4.2, but some that has also made us notice a few common variations of the methodology. Such variations are often due to external factors or project specific preferences that prevent from applying XD exactly in its original form.

One of the first observations made was that most ontology engineering projects are still quite small. There are of course a few large multi-decade efforts, like the Gene Ontology, Cyc, etc., and a number of medium sized ones. However, the large bulk of ontology engineering projects are still run by just a few people over a time period of a few months to a few years, typically as a sub-project of a larger effort, e.g., a software project, or a linked data publishing project. In such a setting an XD configuration with several design pairs working in parallel and a dedicated integration team may not be realistic. It may even be the case that an ontology is created by one individual, e.g., in the context of a larger software project. What adaptations need to be made to XD to be applied in such a setting?

In the projects matching this description that have been observed, XD has very successfully been applied also there. The requirement of pair design may have to be relaxed, and one or two ontology engineers work separately on creating the modules. The advantages of such a setting is that everyone involved has a good overview of the overall ontology at all times, which makes integration easier since problems are anticipated already when designing the modules. However, the disadvantage may be that it is easy to slip back into a mindset where you take on too large challenges at a time, trying to design the overall ontology at once, instead of focusing on one single modelling issue at each point in time. However, also in this case the use of ODPs helps the ontology engineer to keep focused on one single problem at a time, since ODPs are inherently small and modular. Additionally, running XD in “single developer-mode” may have consequences on documentation and testing, since there is no external scrutiny of documentation nor of test setup and results during the development. This makes it even more important to in this setting have frequent contacts with customers and potentially add external reviews to the evaluation phase before the release of an ontology version.

Another very common situation is to have distributed engineering teams. Nowadays many teams are distributed both geographically, and as a consequence may also be highly distributed over time zones. So even if the team is larger than one or two ontology engineers, it may not be feasible to apply pair design as it was originally intended, i.e., two ontology engineers in front of one screen. Two main ways to deal with this issue have been observed; either pair design is completely left out of the methodology, and each ontology engineer works individually, or a compromise is made and engineers still work in pairs but using a more asynchronous method of collaboration, such as sending solution suggestions back and forth or collaborating on the same WebProtégé project but editing it at different points in time. Although the effects of this have not been evaluated in detail, experiences show that despite the shortcomings of the asynchronous communication, it is still worth the effort in terms of improved quality of the resulting model, compared to an ontology engineer working on his or her own.

A further observation is that originally XD was proposed with only two roles of the project participants; customer and ontology engineer. However, in reality, there can be a variety of specialisations of these roles, depending on the char-

acteristics of the project but also on the skills and competencies of the project staff. On the customer side, there is usually at least two types of people; domain experts and software developers. Since ontologies are rarely used in isolation, there is usually someone (other than the ontology engineers themselves) building, or setting up, the software that is going to use the ontology. Whether it is a simple storage solution to serve a SPARQL endpoint to the web, or whether it is a complex AI system to use the ontology for various reasoning tasks, these software developers need to get at least some understanding of how the ontology is being built, what design solutions it contains, and what functionality it supports. Additionally they will have requirements of the ontology, such as what namespaces it uses, how versions are to be handled, typical queries etc. Communication with this kind of “customer” is however very different from the communication that needs to take place with a domain expert, who may not have any technical knowledge at all. Recognising this difference, and involving different kinds of customers in different ways throughout the XD workflow is important, but it is currently not specified in detail in the methodology itself.

Similarly for the ontology engineers of the project, some may be more suited for performing testing, while others may be suited for designing modules or integrating them. The experiences of the authors indicate that the integration loop is the most challenging part of the methodology, where you have the least support from ODPs. Therefore it may be better to assign the most experienced ontology engineers to the integration team, while more inexperienced engineers can do quite well on module development, with the help of a proper ODP catalogue.

Finally, despite the fact that XD is heavily focused on reuse, it does not really detail how to manage other reuse than the selection and application of ODPs. As mentioned earlier, an ontology is rarely constructed completely from scratch. Concerning this, mainly projects that reuse existing ontologies have been observed, e.g., W3C standard for instance, where these have simply been used as a starting point of the overall ontology, and complementing modules have then been created and integrated with that starting point. However, this is also a point of future work for improving XD, to develop more detailed guidelines for integrating existing resources.

2.4.4. ODPs and XD as a Learning Tool

In addition to using ODPs and XD in actual ontology engineering projects, there is also quite an extensive track record of using them for teaching purposes. Experiences of the authors indicate that ODPs are highly suitable as a way of introducing various trade-offs and design choices and their consequences to inexperienced ontology engineers. Participants need some basic knowledge of modelling in general, and of the languages involved, but ODPs then have the potential of creating a fast-track from basic knowledge of OWL to a reasonable set of modelling skills, in order to take part in realistic large-scale ontology engineering projects.

ODPs and XD have been taught in various settings²⁶ over the past decade. While it is commonly necessary to start by introducing the basics, and the basic

²⁶For an (incomplete) list of courses and their content, see: <http://ontologydesignpatterns.org/wiki/Training:Main>

idea behind ODPs, practical modelling using ODPs for modular ontology engineering is then introduced very quickly, and already after 1-2 days of training the participants are usually ready to take on a small “ontology engineering project”. The latter is usually conducted as an XD project, where the participants are divided into pairs and work according to the XD methodology on a shared development project. This both practices their collaboration skills, but also allows the participants to uncover and analyse shortcomings in their own ontology designs. For instance, when students are to integrate modules built by other students, they often realise the importance of documentation, since it is not as easy as they expected to understand someone else’s model. Thereby the authors are convinced that XD is a very suitable methodology to apply when teaching ontology engineering.

2.5. Summary and Future Work

This chapter presented how ODPs can be used in the ontology engineering process, and in particular the XD methodology for ODP-based ontology engineering was introduced. XD is an agile and iterative ontology engineering methodology, that incrementally builds an ontology as a composition of a set of ODP-based ontology modules. Both the use of ODPs as such, and the XD methodology, have been studied in a number of settings, allowing us to confirm a set of positive effects, such as reduced error rate in the resulting ontology, improved documentation and consequently increased understandability of the resulting ontology, as well as a number of subjectively perceived benefits, such as easier and faster development of ontology modules. Applying the XD methodology also allows to have a working increment early in the project, which reduces the risk of misinterpreted requirements, as the (partial) solution can be confirmed and validated in its intended usage scenario very early on. Due to this, XD is also particularly suitable for rapid prototyping of ontologies.

As experiences from working with XD during a number of years show, there are a number of adaptations of XD that have been useful in the realistic settings where it has been applied. In particular, relaxing the criteria of pair design, and using mainly online collaboration instead of face-to-face interaction has worked well in settings where practical constraints have made this the most feasible solution. To further detail the roles of participants, e.g., so that more experienced ontology engineers are assigned specific tasks, such as managing the integration loop, have also proved to be beneficial, rather than to treat all design pairs equal in the project. However, how to more precisely specify the suitable competencies for the various roles and tasks is still future work.

Additional future work consists of providing more detailed guidelines for reusing existing ontologies, and even non-ontological resources, in the XD process, as well as improving the tool support for the methodology. Despite the fact that there exist tools for most tasks in the XD process, only a few of them are integrated into an ontology engineering IDE, and there is still a lack of specific support for the XD workflow in these IDEs. Additionally, since XD relies heavily on the use of ODPs, the result of an XD project may only be as good as the ODPs available and used. In a larger, more long-term ontology engineering project, it is

reasonable to set up your own emerging ODP catalogue, while in smaller projects one usually relies entirely on ODPs available in online catalogues. For this reason, the quality and coverage of such catalogues is a crucial point of improvement in order to apply XD more successfully in smaller projects, especially if there is a lack of ontology engineering expertise in the development team.

Bibliography

- [1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, 1977.
- [2] V. Basili, G. Caldiera, and D. Rombach. The experience factory. In J. Marciniak, editor, *Encyclopedia of Software Engineering*., pages 469–476. Wiley, New York, 1994.
- [3] E. Blomqvist, A. Gangemi, and V. Presutti. Experiments on Pattern-Based Ontology Design. In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*, pages 41–48. ACM, 2009.
- [4] E. Blomqvist, V. Presutti, E. Daga, and A. Gangemi. Experimenting with eXtreme Design. In *Proceedings of EKAW 2010 - Knowledge Engineering and Knowledge Management by the Masses, Lisbon, October 11-15*, LNCS, pages 120–134. Springer, 2010.
- [5] E. Blomqvist, A. Seil Sepour, and V. Presutti. Ontology testing - methodology and tool. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012*, volume 7603 of *LNCS*, pages 216–226. Springer, 2012.
- [6] Z. Dragisic, P. Lambrix, and E. Blomqvist. Integrating ontology debugging and matching into the extreme design methodology. In *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015) Bethlehem, Pennsylvania, USA, October 11, 2015.*, volume 1461. CEUR Workshop proceedings, 2015.
- [7] A. Gangemi and V. Presutti. Ontology Design Patterns. In *Handbook on Ontologies, 2nd Ed.*, Int Handbooks on Information Systems. Springer, 2009.
- [8] M. Gruninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.
- [9] K. Hammar. Ontology design patterns in use - lessons learnt from an ontology engineering case. In *WOP 2012: Proceedings of the 3rd Workshop on Ontology Patterns, in conjunction with the 11th International Semantic Web Conference (ISWC) 2012, CEUR Workshop Proceedings*, volume 929. CEUR Workshop proceedings, 2012.
- [10] K. Hammar. Ontology design pattern property specialisation strategies. In *Knowledge Engineering and Knowledge Management - 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings*, volume 8876 of *LNCS*, pages 165–180. Springer, 2014.

- [11] K. Hammar. Ontology design patterns: Improving findability and composition. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, volume 8798 of *LNCS*, pages 3–13. Springer, 2014.
- [12] K. Hammar. Ontology Design Patterns in WebProtégé. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015)*, volume 1486. CEUR Workshop proceedings, 2015.
- [13] L. Iannone, A. Rector, and R. Stevens. Embedding knowledge patterns into owl. In *The Semantic Web: Research and Applications - 6th European Semantic Web Conference, ESWC 2009 Heraklion, Crete, Greece, May 31–June 4, 2009 Proceedings*, volume 5554 of *LNCS*. Springer, 2009.
- [14] C. M. Keet and A. Lawrynowicz. Test-driven development of ontologies. In *The Semantic Web. Latest Advances and New Domains 13th European Semantic Web Conference, ESWC 2016, Anissaras, Crete, Greece, May 29 – June 2, 2016. Proceedings*. Springer, 2016.
- [15] V. Presutti, E. Bomqvist, E. Daga, and A. Gangemi. Pattern-based ontology design. In *Ontology Engineering in a Networked World*, pages 35–64. Springer, 2012.
- [16] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme Design with Content Ontology Design Patterns. In *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009*, volume 516. CEUR Workshop proceedings, 2009.
- [17] J. Shore and S. Warden. *The art of agile development*. O'Reilly, 2007.
- [18] O. Zamazal and V. Svatek. Patomat – versatile framework for pattern-based ontology transformation. *Computing and Informatics [online]*, 4(2):305–336, 2015.

Chapter 3

Quality of Content Ontology Design Patterns

Karl Hammar, Jönköping University, Linköping University

3.1. Introduction

The idea of using Ontology Design Patterns, particularly Content Ontology Design Patterns (hereafter CODPs) to support Ontology Engineering has gained traction within the academic community, as evidenced by the Workshop on Ontology Patterns series of workshops held in conjunction with the International Semantic Web Conference, and the recent formation of the Association for Ontology Design & Patterns (ODPA). Adopting CODPs and the associated eXtreme Design [24] (hereafter XD) Ontology Engineering method allow developers to reuse known good solutions to modelling problems, and thus hopefully enables these developers to work in a more efficient manner.

However, as the author has previously shown [14], the published work on Ontology Design Patterns is lacking in some aspects. While many patterns have been presented, not enough work has been done on evaluating the structure and design of those patterns. Consequently, little is known about what qualities or properties of patterns are beneficial in ontology engineering tasks, and inversely, what properties are not helpful or are possibly even harmful in such tasks.

This chapter summarises the author's recent work on remedying the aforementioned lack of knowledge, by developing a CODP Quality Model. This work is important for three main reasons:

Firstly, if Semantic Web technologies are to reach broader adoption outside of academia, the academic community needs to present a tenable value proposition – we need to offer mature methods, patterns, and tooling that show industry clear benefits over alternative approaches. Given the relative newness of Semantic Web technology and the complexity of the standards and tool stack, both CODPs and CODP tooling need to be of sufficient quality that they can be used easily by non-expert ontologists, i.e., not only academic prototypes. We need to understand and improve their quality.

Secondly, there are several notions with regard to CODP structure and quality that are assumed, but not evaluated. For instance, it is generally agreed that CODPs should be small – but is this always true? And if so, how small is small? Or it is assumed that the shared conceptualisation brought about by the many CODPs that reuse ideas from the DOLCE ontology provides a degree of interoperability which is beneficial – but does this not constrain the use of those CODPs? Providing a framework for how to speak of and measure quality allows the CODP community to test these assumed hypotheses and possibly break preconceived notions.

Thirdly, as illustrated quite clearly by the example given in the previous paragraph, there are several cases where CODP qualities clash with one another. Another common case of such clashes are between reasoning performance efficiency on the one hand and full logic expressiveness on the other. This is of course not news to Semantic Web researchers, but it is important that we understand these clashes and the tradeoffs they give rise to, when pitching CODPs as a solution to industry or to researchers from other fields.

The chapter is organised as follows: Section 3.2 summarises existing work on quality evaluation for ontologies, models, and software, upon which the latter sections extend. Section 3.3 introduces and describes the CODP Quality Model, consisting of a set of quality characteristics and quality indicators relevant to evaluate CODPs by. Section 3.4 suggests areas for further work in CODP quality research, exemplifies some tensions between qualities and discusses the resulting tradeoffs to be made when designing and using CODPs, and recommends improvements to CODP tools based on the work presented here, before Section 3.5 closes the chapter.

3.2. Related Work

The following section presents existing approaches to quality evaluation which have been reused in the construction of the CODP Quality Model. Section 3.2.1 discusses evaluation and quality of related non-ontological resources (information systems, ER models, and OOP patterns), while Section 3.2.2 discusses related work on ontology evaluation.

3.2.1. Quality in Related Fields

3.2.1.1. Information System Quality

Ontologies are almost always used as components within an information system. As such, research on software artefact quality is useful in understanding the demands on an ontology from an information system perspective. This field has seen considerable work going back to the 1970s. In particular, the ISO standard 25010 [16] introduces a *Product Quality Model* which covers many aspects of quality that are transferable to a CODP context.

The ISO 25010 Product Quality Model is expressed according to a framework that defines certain basic concepts. Per this framework Quality Characteristics,

Table 3.1. ISO 25010 Product Quality Model (adapted from [16])

Qual. characteristic	Sub-Characteristic	Qual. characteristic	Sub-Characteristic
Func. suitability	Functional completeness Functional correctness Functional appropriateness	Reliability	Maturity Availability Fault tolerance Recoverability
Perf. efficiency	Time behaviour Resource utilisation Capacity	Security	Confidentiality Integrity Non-repudiation
Compatibility	Co-existence Interoperability		Accountability Authenticity
Usability	Appropriateness recognisability Learnability Operability User error protection User interface aesthetics Accessibility	Maintainability	Modularity Reusability Analysability Modifiability Testability
Portability	Adaptability Installability Replaceability		

Quality Properties, and Quality Measures are disjoint but related concepts. Quality Characteristics are general and not directly measurable attributes of an artefact, such as *Usability* or *Reliability* (Quality Characteristics may also have more specific Sub-Characteristics). Quality Properties are measurable attributes of an artefact that contribute to some Quality Characteristics, and Quality Measures are the results of performing measurement of such properties by some method.

The Product Quality Model [16] is displayed in Table 3.1. It defines a set of eight quality characteristics, each composed of two to six sub-characteristics. These quality characteristics have definitions written from a system or product perspective. For instance, the quality characteristic *availability* is defined as *degree to which a system, product or component is operational and accessible when required for use*. The quality characteristics have been adapted for an ontology and CODP context and are used as the basis for the set of quality characteristics included in the CODP Quality Model (presented in Section 3.3.1).

While ISO 25010 also states the existence of measurable quality properties contributing to these quality characteristics, no instances of such quality properties are introduced or formalised.

3.2.1.2. ER Model Quality

In [6] Genero et al. study a set of metrics believed to affect the learnability and modifiability of ER models. Their study participants were given a set of ER models that differed in the metrics being studied, and were tasked with first filling out a questionnaire to evaluate their understanding of the ER model, and secondly, to modify the ER model in accordance with a newly introduced set of additional requirements. In analysis, the time taken to respond to the questionnaire and to perform the model changes was studied to ascertain the relative understandability and modifiability of the ER models that exhibited different values for the metrics of study. Genero et al. find a significant correlation between a higher number of

relations and attributes within the model and an increased understandability and modifiability time, but no significant correlation between the number of entities as a whole and understandability.

Moody and Shanks [22] discuss the issue of redundancy and simplicity, arguing that simpler models have shown to be more flexible, easier to implement, and easier to understand. They suggest that if the size of a model is calculated from the number of entities and relationships in the model, the simplest solution is the one that minimises this size. However, it is important to note that the model must still be suitable for its intended purpose. This mirrors the perspectives brought forward by Lindland et al. in [18]: a high-quality model is constrained by both completeness and relevance criteria, such that it should be large enough to fulfil its purpose, but no larger than that.

In the CODP Quality Model the work presented by Moody and Shanks, and Lindland et al., are captured in quality indicators measuring CODP size and CODP minimalism (in Section 3.3.2 listed as *MI10* and *MI20*), contributing to the learnability and operability of CODPs. The work presented by Genero et al. (both their findings and method) are also reused – the findings in the form of indicators measuring class to property ratios (*MI07*) contributing to learnability and operability, and the method in the form of in-use indicators of usability and modifiability (*UI01* and *UI02*).

3.2.1.3. Pattern Quality

Ontology Design Patterns are a sort of design patterns, that is, packaged solutions to commonly occurring problems. In this, they share purpose with software design patterns, the most common of which are design patterns for object oriented programming (OOP). Prechelt et al. [23] report on two experiments in which the characteristics of OOP design pattern code implementations (in particular, the number of pattern comment lines, PCL, associated with each such implementation) affect the maintainability of software products in which the patterns are used. They find that the more well documented and explicit that design pattern usage in software code is, i.e., the higher the PCL value, the faster and better (in terms of rarity of errors made) maintenance tasks are performed over that software code. This finding is mirrored in the CODP Quality Model via an indicator measuring the annotation coverage of the CODP (*MI01*).

3.2.2. Ontology Quality

One perspective on CODPs is to consider them as being simple, small, and reusable ontologies. The following section introduces work on ontology evaluation frameworks, methods, and indicators, that have been studied during and in several cases influenced development of the CODP quality model. In addition to the ontology evaluation work introduced below, the interested reader is referred to [7], in which Goméz-Peréz et al. summarise and discuss several types of common taxonomical errors and anti-patterns in ontologies.

3.2.2.1. O² and oQual

A thorough study on the evaluation of ontologies is performed by Gangemi et al. in [3] and [4]. Their approach is based on two perspectives of ontologies, formalised into two meta-ontologies for understanding, classifying, and selecting ontologies, O^2 and *oQual*.

The O^2 meta-ontology views ontologies as semiotic objects, i.e., information objects with intended conceptualisations to be used in particular communication settings. O^2 holds concepts such as *Rational agent*, *Conceptualisation*, and *Graph* – representing the settings in which ontologies are used, the users of said ontologies, their intended meaning of the ontologies, the actual implementation (i.e., graph models), etc. This model also holds the concept *QOOD*, or *Quality Oriented Ontology Description*, which represents roles and tasks associated with processes and elements of the ontology – in essence, a type of requirements specification for part of, or a whole, ontology engineering project.

Based on the concepts in O^2 , three dimensions of quality or evaluation are presented and discussed – *structure*, *functionality*, and *usability*. The first kind of measures treat the ontology as a directed graph (which is consistent with the RDF data model) and concern the structures present in this a graph. This includes measures like subsumption hierarchy depth, breadth, fan-out, cycle ratios, density, etc. Several of these measures have been reused as indicators in the CODP Quality Model, including the notions of axiom to class ratio, class disjointness ratio, class to property ratio, and tangledness (in Section 3.3.2 listed as indicators *MI05*, *MI06*, *MI07*, and *MI23*). The second kind of measures concern the intended functionality of the ontology, and include such examples as precision, coverage, and accuracy, which are all measured against some set of requirements over the ontology. The third kind of measures concern the communication aspects of the ontology, i.e., how it is documented, annotated, and understood by users. This includes measures related to recognition, efficiency, and interfacing. While the structural measures included in [4] are presented using formal definitions (in many cases even mathematic formulas) the functionality measures are less formally defined (giving some specific indicators, but mostly general method suggestions), and usability measures are even less defined (given almost entirely as examples or areas for future development).

The *oQual* formal model for ontology validation (and its associated meta-ontology) provides a bridge connecting the ontology engineering situation and context (modelled according to O^2) with the aforementioned measures, enabling validation that a certain developed ontology is sufficient and appropriate for the use for which it was developed. *oQual* includes concepts like value spaces and parameters over ontology elements, ordering functions for selecting parameters from different QOODs to prioritise, trade-offs that may need to be made, etc.

3.2.2.2. ONTOMETRIC

ONTOMETRIC, introduced in [21], is an approach for formalising ontology suitability for different tasks, heavily influenced by the Analytic Hierarchy Process (AHP) [25], an established method for aiding decision-making when dealing with multi-criteria problems. Per this method, a multi-criterion problem is broken

down into the different criteria that need to be met. These criteria are sorted using a comparison matrix, such that the “competing” criteria on each level of the decision hierarchy are easily and intuitively compared pairwise, and the resulting prioritisation used to calculate a weighting of the total set of criteria with respect to the problem. When selecting between available alternatives, the weighting can be used to calculate a total suitability score for each option (i.e., the highest score would be associated with the most suitable ontology for reuse).

In order to support this method, ONTOMETRIC provides a set of criteria for ontology suitability that can be used to populate an AHP decision hierarchy. These general criteria of ontology suitability are divided up into five different dimensions, representing different aspects of an ontology suitability problem: content, language, methodology, tool, and costs. A drawback of using ONTOMETRIC criteria for quality assurance purposes is that the method and model does not in itself suggest which criteria are useful in which situations - it merely provides them as examples of things that can be measured and prioritised by the user. Added to that, the criteria are rather technical and would likely be inaccessible to the non-expert user. However, for the intended usage, ontology selection guidance for ontology engineers), particularly in the early phases of an ontology engineering project, ONTOMETRIC has great potential.

3.2.2.3. OntoClean

One of the most well-known methodologies for evaluating the conceptual consistency of ontologies is OntoClean by Guarino and Welty [8, 9, 26]. OntoClean uses a logical framework including very general ground notions and definitions from philosophy, believed to hold in any reasonable representation of the world, including an ontology model. The framework includes the notions of *rigidity*, *identity*, and *unity* as characteristics applicable to classes in an ontology, and a set of constraints on which taxonomic relations that may exist between classes that exhibit these different characteristics. The listed characteristics are in OntoClean parlance denoted *metaproPERTIES*.

By annotating the classes in an ontology using the OntoClean metaproPERTIES and checking whether the associated constraints hold, a developer can test whether their ontology is conceptually and philosophically consistent with regards to the notions of rigidity, identity and unity. It should be noted that this is not a guarantee that the ontology is sound with respect to real world phenomena or requirements. The methodology has been applied beneficially in several projects [2, 5].

3.3. The CODP Quality Model

Based on the related work presented in Section 3.2 and extending on the work detailed in [12], the author has developed a CODP Quality Model, consisting of three main components:

1. A CODP Quality Metamodel.

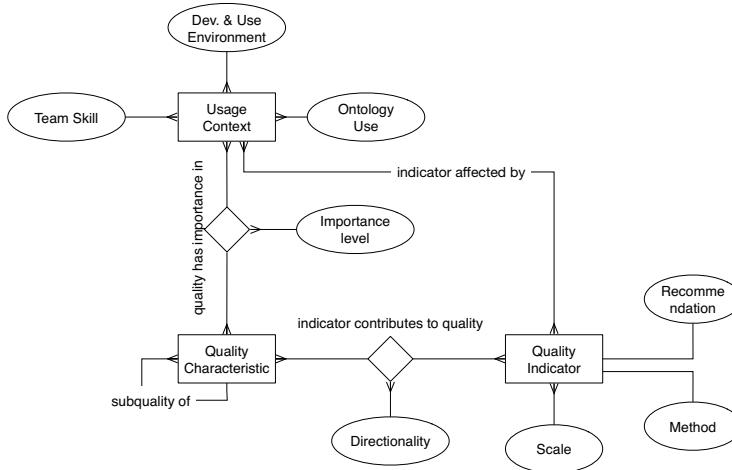


Figure 3.1. CODP Quality Metamodel

2. A set of CODP quality characteristics encapsulating different aspects of quality as relevant to CODP usage.
3. A set of CODP quality indicators that measure to what degree a given CODP expresses the aforementioned quality characteristics.

The quality characteristics and quality indicators are discussed in Sections 3.3.1 and 3.3.2 respectively. The notions of quality characteristic and quality indicator are defined in the CODP Quality Metamodel, which provides a conceptual understanding of how to discuss quality as applied to CODPs. The CODP Metamodel is illustrated in Figure 3.1. The key concepts it encompasses are:

- **Usage Context** – Represents the context in which a CODP is used to construct an ontology. Component concepts of this context are the skills of the ontology engineer(s), the use to which the resulting ontology is intended to be put, and the social or business environment in which the development and subsequent use of the ontology will take place (the latter including aspects such as development method, team distribution, customer relationship, etc).
- **Quality characteristic** – Denotes a particular aspect of CODP quality that may be of greater or lesser importance in a particular usage context. Quality characteristics can be decomposed into sub-characteristics. Quality characteristics represent concerns or perspectives on quality on an abstract level. They are not themselves directly measurable using some metric or method.
- **Indicator** – Individually measurable properties of an CODP that contribute to increasing or decreasing some quality characteristic(s). Can be accompanied by scales of measure, measurement methods, and recom-

Table 3.2. CODP Quality Characteristics. For indicator names, see Table 3.3.

Quality characteristic	Sub-Characteristic	Indicators
Functional Suitability	Functional Completeness Functional Appropriateness Consistency Accuracy	MI12
Usability	Appropriateness recognisability Learnability Operability User error protection User interface aesthetics Accessibility	DI01, DI03, DI06, DI07 DI03-DI07, MI01, MI02, MI07, MI10, MI17-MI23, UI01, UI03 MI02, MI07, MI10, MI19, MI21-MI23, MI25 DI02 DI04 DI01, DI06
Maintainability	Modularity Analysability Modifiability Testability Stability	MI17, MI18 MI01, MI05, MI20 UI02
Compatibility	Reusability Co-existence Interoperability	MI17, MI18, MI25 MI10 MI25
Resulting performance efficiency		MI02-MI04, MI06, MI08, MI09, MI11, MI13-MI19, MI22-MI25

mended values. Can be affected by usage context.

It should be noted that the CODP Quality Model is partially (e.g., [10–13]) but not exhaustively evaluated. The author invites the Semantic Web research community to further test the model’s indicators and their predicted effects.

3.3.1. Quality Characteristics

The CODP Quality Model quality characteristics (developed from the ISO 25010 Product Quality Model discussed in Section 3.2.1.1) are presented in Table 3.2, and discussed in detail below. Table 3.2 also includes references to the quality indicators that contribute to each quality characteristic.

While most of the developed quality characteristics are associated with one or more quality indicators, several are not. This does not imply that those characteristics are unimportant, but rather that more work is required in order to find suitable indicators to measure them.

3.3.1.1. Functional Suitability

Degree to which a CODP meets stated or implied needs.

- *Functional completeness* – Degree to which the CODP meets expressed knowledge modelling requirements (i.e., competency questions and other design requirements).
- *Functional appropriateness* – Degree to which the CODP facilitates simple storage and retrieval of knowledge formalised according to its definitions

(e.g., does the CODP require simple or complex SPARQL queries to retrieve knowledge).

- *Consistency* – Degree to which the CODP is internally logically consistent.
- *Accuracy* – Degree to which the CODP accurately represents the real world domain being modelled (e.g., whether it adheres to established industry standards and protocols, or legislation).

3.3.1.2. Usability

Degree to which a CODP can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction.

- *Appropriateness recognisability* – Degree to which users can recognise whether a CODP is appropriate for their needs.
- *Learnability* – Degree to which a CODP's structure, and intended usage can be learned by users new to it, such that they can thereafter apply the CODP successfully and efficiently.
- *Operability* – Degree to which a CODP has attributes that make it easy to apply and use.
- *User error protection* – Degree to which a CODP prevents users from making modelling errors.
- *User interface aesthetics* – Degree to which the CODP's documentation (text, graphics, etc.) is pleasing for the user.
- *Accessibility* – Degree to which the CODP's documentation can be used by people with the widest range of characteristics and capabilities.

3.3.1.3. Maintainability

Degree of effectiveness and efficiency with which a CODP (and consequently, ontologies built using that CODP) can be adapted and modified by maintainers after deployment in some usage scenario.

- *Modularity* – Degree to which the CODP is composed of discrete components such that a change to one component has minimal impact on other components.
- *Analysability* – Degree of effectiveness and efficiency with which it is possible to assess the impact on a CODP-based ontology module (or the CODP itself) of intended change to one or more of its parts, or to diagnose a CODP for deficiencies or causes of failures, or to identify parts to be modified.
- *Modifiability* – Degree to which a CODP-based ontology module (or the CODP itself) can be effectively and efficiently modified without introducing defects or degrading existing CODP quality.
- *Testability* – Degree of effectiveness and efficiency with which test criteria can be established for a CODP and tests can be performed to determine whether those criteria have been met.
- *Stability* – Perceived change expectation on the CODP - high stability denotes a low degree of change is expected, and vice versa.

3.3.1.4. Compatibility

Degree to which a CODP can successfully be reused and integrated with other CODPs or IT artefacts in the construction of ontologies or systems.

- *Reusability* – Degree to which a CODP can be used in more than one system, or in building other assets.
- *Co-existence* – Degree to which a CODP can coexist with other CODPs as modules in an ontology, i.e., without detrimental impact on other CODP modules.
- *Interoperability* – Degree to which two or more CODPs share representations of co-occurring concepts.

3.3.1.5. Resulting performance efficiency

Reasoner or system performance efficiency over ontologies created using the CODP. Can be discussed in terms of the average time taken, or of the system resources required, to materialise inferences of ontologies using this CODP.

3.3.2. Quality Indicators

Table 3.3 lists a set of 36 quality indicators that measure different aspects of CODP quality. The indicators are in the table and in the subsequent discussion grouped by the artefact or action that they most directly measure - the CODP documentation, the CODP model, or the CODP when used by human ontology engineers:

- The *documentation quality indicators* all concern the textual and graphical description of a CODP, typically provided in the form of an accompanying web page or other document. Such indicators contribute mostly to usability-related CODP qualities, e.g., appropriateness recognisability, learnability, operability, etc.
- *Model indicators* measure different aspects of the reusable OWL building block that makes up the core of a CODP, or its underlying RDF/RDFS model. These indicators contribute to qualities that are important in terms of how a CODP or an CODP-based ontology is used together with other CODPs, ontologies, or in information systems, including maintainability, compatibility, and reasoning performance.
- *In-use indicators* measure properties of CODPs that cannot be captured using quantitative methods over the CODP itself, but rather requires the use or evaluation of that CODP by a human ontology engineer. These indicators contribute primarily to those qualities that are contextually bounded, e.g., maintainability, usability, and functional suitability.

The indicators, their provenance, and effects, are discussed in the following sections. For reasons of brevity, not all of the indicators in Table 3.3 are covered. The interested reader can find more details, including suggested measurement methods and scales for several indicators, in [12].

Table 3.3. CODP Quality Indicators

Category	Nr	Indicator
DOCUMENTATION INDICATORS		
	DI01	Accompanying Text Description
	DI02	Common Pitfalls Description
	DI03	Competency Question Count
	DI04	Documentation Minimalism
	DI05	Example Illustration Count
	DI06	Example Text Count
	DI07	Structure Illustration
MODEL INDICATORS		
	MI01	Annotation Ratio
	MI02	Anonymous Class Count
	MI03	Average Class In-Degree
	MI04	Average Class Out-Degree
	MI05	Axiom/Class Ratio
	MI06	Class Disjointness Ratio
	MI07	Class/Property Ratio
	MI08	Cyclomatic Complexity
	MI09	Existential Quantification Count
	MI10	Minimalism
	MI11	Nary Relation Count
	MI12	OntoClean Adherence
	MI13	OWL Horst Adherence
	MI14	OWL 2 EL Adherence
	MI15	OWL 2 QL Adherence
	MI16	OWL 2 RL Adherence
	MI17	Property Domain Restrictions
	MI18	Property Range Restrictions
	MI19	Redundant Axiom Count
	MI20	Size
	MI21	Subsumption Hierarchy Breadth
	MI22	Subsumption Hierarchy Depth
	MI23	Tangledness
	MI24	Terminological Cycle Count
	MI25	Transitive Import Count
USAGE INDICATORS		
	UI01	Functionality Questionnaire Time
	UI02	Modification Task Time
	UI03	User-Reported Abstraction Level Rating

3.3.3. Documentation Indicators

The first documentation indicator, *DI01 – Accompanying Text Description*, simply measures whether there actually exists a dedicated documentation page or other equivalent documentation for a given CODP module. It may be surprising to the reader that CODPs exist that are not at all documented. This can occur when patterns are extracted from existing ontologies or other data structures in an exploratory manner, as opposed to designed in a structured manner, particularly in project contexts where the resulting CODPs are intended to be used directly and not necessarily published for a wider audience. Examples of such undocumented CODPs¹ include Nary Classification² and Time Indexed Membership³.

¹At the time of writing, 2016-03-29

²<http://www.ontologydesignpatterns.org/cp/owl/naryclassification.owl>

³<http://ontologydesignpatterns.org/cp/owl/timeindexedmembership.owl>

Given that CODP documentation exists, a variety of usability aspects stem from the size and complexity of this documentation. It is important to provide the reader with all the information they need to ascertain the utility and intended use of an CODP, while at the same time not overwhelming them with too much or unnecessary information. Lodhi and Ahmed [19] find that several of the documentation fields used in the OntologyDesignPatterns.org portal are by users and maintainers consistently ranked more important to understanding the CODP than others. The quality indicator *DI04 – Documentation Minimalism* captures the need to display all of this core documentation, and no more. Per this indicator, the documentation fields shown should primarily include graphical representation, examples, pattern intent, OWL building block link, OWL example file, competency questions, common pitfalls, and consequences of use – other documentation fields should be hidden unless the user specifically asks for them. Initial evaluation indicates that documentation minimalism indeed correlates with faster understanding of the CODP by non-expert users [12].

Of the above listed description component fields, the author's forthcoming work indicates that the most important ones affecting appropriateness recognisability are, in order, examples of use (measurable by *DI05 – Example Illustration Count* and *DI06 – Example Text Count*), description of intent, competency questions (measurable by *DI03 – Competency Question Count*), and graphical representation (measurable by *DI07 – Structure Illustration*). The component field “description of intent” is not associated with an indicator of its own due to the difficulty in quantifying such an indicator, and due to the overlap with the aforementioned indicator *DI04* which anyway requires it.

In addition to the count of illustrations of structure (*DI07*) and example uses (*DI05*), it is reasonable to assume that the format and design of those illustrations also have usability-related effects. Some results of the authors ongoing work, include that users report preferring graph-oriented representations over tree-oriented representations or Venn diagrams. When presented with several common graphical OWL representations, the VOWL format⁴ [20] was consistently the most preferred.

3.3.4. Model Indicators

CODPs are intended to be reusable, from which follows that they should be able to represent or solve the modelling issue for which they are intended to be used, but make few (ideally no) ontological commitments not pertaining to this issue – i.e., they should contain no extraneous axioms not required to fulfil their design requirements. The measure of such CODP minimalism is captured in the indicator *MI10 – Minimalism*, which contributes to increased learnability and operability of a CODP (this effect is also apparent in other models than CODPs, see the discussion in Section 3.2.1). Minimalism also contributes to the co-existence quality characteristic, i.e., the ability of the CODP to co-exist with other CODPs in an ontology without detrimental effects.

Since the OWL imports feature is transitive and complete, and since CODPs often build upon and specialise one another, relatively few CODPs are actually

⁴<http://vowl.visualdataweb.org/>

minimal. It is generally the case that several foundational concepts and properties, that are not strictly necessary to the modelling scenario at hand, are included in an CODP through imports. The number of ontologies or CODPs in the transitive import closure of an CODP is measured via the indicator *MI25 – Transitive Import Count*, which contributes negatively to operability – since more work steps are needed to reconcile these foundational and many times not strictly needed concepts with a model under development. The same indicator also affects interoperability, increasing it for those pairs of CODPs that share foundational concepts, but decreasing it for those that do not.

A CODP that expresses the minimalism property may still be difficult to understand and operate due to its size (i.e., even though the CODP does not exceed its design requirements, those design requirements are so large as to be unintuitive to fulfil in one single CODP). CODP size can be measured in a variety of ways, by number of axioms, number of classes, number of RDF triples, number of bytes in some given serialisation format, etc. The indicator *MI20 – Size* does not specify which measure must be used, but recommends that OWL-level constructs be used as its basis, since that is the level that ontology development tooling tends to operate on, and which users see. The author's prior work [10] indicates that an appropriate size for users to understand fully within a minute or two of study is three to four classes, and the properties and restrictions associated with them. If CODPs are significantly larger than this they could be considered candidates for splitting apart into sub-patterns, particularly if intended to be used by non-expert ontology engineers.

The structure of a CODP's class subsumption hierarchy can have significant effect on the subsumption hierarchy of an ontology constructed using that CODP. Indicators that measure these structures include *MI21 – Subsumption Hierarchy Breadth*, *MI22 – Subsumption Hierarchy Depth*, and *MI23 – Tangledness*. The former two measure the average number of siblings on a given subsumption level of the CODP, and the average depth in number of levels of subsumption respectively. The latter measures the number of multi-parent classes. Higher values for either of these indicators affect learnability and operability of an CODP negatively [3]. The latter two have also been shown to contribute negatively to reasoning performance [17]. It should be noted that while the mentioned effects on usability-related qualities are derived from the asserted subsumption hierarchy only, the reasoning effects are of course also derived from the inferred subsumption hierarchy, including anonymous classes.

There are several quality indicators that measure properties of the underlying RDF representation upon which an OWL CODP is built. These indicators include *MI03 – Average Class In-Degree* and *MI04 – Average Class Out-Degree*, which measure the average number of in-going and out-going edges from the set of nodes in the RDF graph that are OWL classes (or in other words, the average number of RDF triples involving OWL classes where those OWL classes are in the subject or object positions). They also include *MI08 – Cyclomatic Complexity*, which measures the number of linearly independent paths through in RDF representation of the CODP. All three of these indicators contribute negatively to reasoning performance efficiency [17].

As shown in [13], object properties in CODPs can be specialised by two dif-

ferent methods, the choice of which affects both the reasoning characteristics and the usability of the resulting ontology or CODP. Either new sub-properties are constructed, with defined domains and ranges corresponding to some specialised classes, or existing properties of the CODP are reused and locally constrained by way of property restrictions on those specialised classes. The former method is preferable in terms of reasoning performance of the resulting ontologies/CODPs, whereas the latter method is preferable for data integration purposes. The indicators *MI17 – Property Domain Restrictions*, *MI18 – Property Range Restrictions*, and *MI09 – Existential Quantification Count* all measure how property usage is constrained in the CODP. Constraining property use in the resulting model, whether by use of domain and range restrictions or anonymous class restrictions (measurable by indicator *MI02 – Anonymous Class Count*), has been shown to improve learnability of the resulting CODP, by providing users with clarification on the intended use of the properties in question [12]. However, the flip side of this is that such constraints limit the reusability of the properties in question, and also increase interdependency between the properties and class definitions that they are constrained by, contributing negatively to CODP modularity.

The indicators *MI05 – Axiom/Class Ratio* and *MI07 – Class/Property Ratio* reflect different aspects of how classes are expressed in the CODP. The former indicator captures a perspective on the logic complexity of an CODP, i.e., whether it is a simple schema with few details described per class, or whether concepts are formalised in a logically more expressive or richer manner. A higher value is preferable in terms of analysability of classes [3]. The latter indicator measures the ratio of classes to properties, i.e., possible interconnections between class members. As discussed in Section 3.2.1, ER models displaying a higher number of relations and attributes per entity are more difficult to understand. The same is true for CODPs; a higher number of properties per class contributes to decreased learnability and operability while the opposite (i.e., a higher value for *MI07*) increases these qualities [3, 12].

Finally, *MI12 – OntoClean Adherence* measures to which degree the CODP is consistent with the OntoClean [9] taxonomic constraints, i.e., whether it is logically sound and consistent vis-a-vis the real world. This indicator contributes to the accuracy quality characteristic. As OntoClean can be time consuming to apply over a larger model, the indicator can be expressed either in terms of complete adherence (is the entire model and all taxonomic relations in it fully consistent according to OntoClean), or in terms of the ratio of inconsistent to consistent taxonomic relations. Most CODPs are small enough that the former measure is feasible to use in terms of size - however, CODPs are often also relatively abstract in nature, which can complicate the tagging of their concepts with OntoClean metaproPERTIES, which is made simpler if those concepts are more tangible.

3.3.5. In-Use Indicators

Indicators *UI01 – Functionality Questionnaire Time* and *UI02 – Modification Task Time* measure the time needed for representative users to perform tasks that require correctly understanding and using a CODP (inspired by the work of

Genero et al. [6] discussed in Section 3.2.1).

In the former case, users respond to a survey on the functionality and intended use of the CODP – for instance, the users might be asked which out of a number of listed competency questions the CODP would be capable of answering, or which class in the CODP that corresponds to a certain term or concept in a textual description. In the case that the same users take multiple surveys for different CODPs, it is important to ensure sufficient randomisation in survey ordering to avoid learning effects affecting the results.

In the latter case, users are tasked with performing a set of modification tasks for a CODP, that require not only understanding the intended CODP usage and structure, but also putting that understanding into practice. Again, in the case that the same users are used to evaluate multiple CODPs, some randomisation of order would be required.

In either case, only the time taken by users who provide correct responses and modification results should be taken into account. The greater the time required, the lower the learnability or operability of the CODP in question. It is however important to note that these indicators are very much dependent on the ontology engineering expertise, or lack thereof, of the participating users – so while they can be useful in evaluating candidate CODPs for a particular project (where the participating rating users are representative of project members), they should not be used to compare CODPs in general.

UI03 – User-Reported Abstraction Level Rating attempts to capture the notion of how abstract or concrete a CODP is. A more abstract CODP (as reported by users) has been shown to be more difficult to understand [12]. Unlike the previous two indicators, abstraction level is difficult to measure in a quantitative manner – we need to rely on user ratings. A survey format using a five-point Likert scale ranging from “Very concrete” to “Very abstract” has worked well for the author in the past. The users doing the rating need to have had sufficient time to study and apply several CODPs in test scenarios before rating them. Providing the users with multiple patterns allows them to see a variation of patterns that makes it easier for them to answer confidently (especially if they lack prior experience of CODPs). As a benchmark of rating reliability, we recommend including some CODPs that have previously been rated a sufficiently large number of times that skew to either end of the spectrum can be identified and adjusted.

3.4. Discussion and Open Questions

In developing the CODP Quality Model the author has made several observations that warrant further discussion. This section provides input to such discussion by suggesting areas for further work in CODP quality, exemplifying tensions and tradeoffs for developers to keep in mind, and recommending CODP tooling improvements taking the presented qualities and indicators into account.

3.4.1. Where are the Gaps?

The CODP Quality Model contains several quality indicators contributing to *Usability* and *Resulting Performance Efficiency*, but fewer contributing to *Func-*

tional Suitability, Maintainability, and Compatibility. This lack is likely a consequence of the difficulty in studying the latter quality characteristics. Understanding *Maintainability* requires studying a real ontology engineering project as it progresses beyond the initial development phases and into deployment and maintenance phases. By contrast, most research projects are formulated to focus primarily on the development phase, and consequently we do not have enough studies on deployment and maintenance yet. *Functional Suitability* is difficult to study rigorously as it is a very context-bounded quality characteristic, making it quite difficult to generalise any findings. Studying CODP *Compatibility* requires that a sufficient number of high quality CODPs from different sources or designed in different styles already exist, that compatibility can be gauged against. This has until recently not been the case.

The lack of indicators for the above mentioned quality characteristics is particularly unfortunate as these characteristics have several sub-qualities that are important in enabling CODP adoption outside of academia. *Maintainability* is perhaps the most obvious candidate for improvement – without understanding how CODP and XD use in ontology engineering affects the need for maintenance and support of developed ontologies, we will not see industry deploying these methods to any greater extent. In particular CODP *Testability* and *Analysability* are critical qualities to understand when developing CODP-based ontologies that will need to be maintained (possibly by another team than the initial developers) in the future. *Testability* is also highly relevant in supporting the XD method, which depends on tests to ensure that component modules of a ontology project do not break during integration and refactoring [24]. In addition to understanding these quality characteristics, tooling to aid in constructing and executing tests, or in analysing CODP-based ontologies, will also need be developed.

3.4.2. Tradeoffs to be Made

As the attentive reader is sure to have noticed, there are indicators that contribute negatively to some quality characteristics while contributing positively to others. The presence of such tensions imply that ontology engineers developing or using CODPs may need to make tradeoffs between which qualities they consider most important in their project. Organised by quality characteristic, the tradeoffs found so far (others may surface as the above mentioned gaps are filled) include *Resulting Performance Efficiency* versus *Functional Suitability*, *Resulting Performance Efficiency* versus *Learnability*, *Interoperability* versus *Operability*, and *Learnability* versus *Reusability*.

The tradeoff between *Resulting Performance Efficiency* and *Functional Suitability* stems from the fact that employing the full logic expressivity of the OWL language is computationally expensive, hence the development of reasoning-friendly OWL subsets and the OWL 2 profiles (captured in indicators MI13-MI16). The most reasoning-efficient OWL model imaginable would be a null model that makes no assertions whatsoever. As a model comes closer and closer to representing some real world domain (i.e., the model's functional completeness increases), the number of axioms (and most likely also the variance of OWL constructs used) in it also increases, which in turn decreases reasoning performance over the model.

One concrete example of this is the use of existential quantification axioms (indicator MI09) which may well be required to accurately model the domain, but which are associated with poor reasoning performance [17]. Another example is the use of property domain or range restriction axioms (MI17 and MI18), which on RDF-level increase the number of ingoing edges to each class definition, which has also been shown to be associated with decreased reasoning performance [17].

When applying CODPs in scenarios where the resulting ontology will be used for reasoning *Reasoning* and where the time or resource consumption characteristics of that reasoning matters, developers are recommended to consider carefully whether the CODPs they use are compliant with their reasoning requirements. The simplest way of going about this is to ensure that the CODPs used comply to a suitable OWL 2 profile. However, it is important to also ensure that in the process of specialising and applying those CODPs, no OWL constructs not supported by the chosen profile are introduced. Thankfully the two common ontology development environments Protégé and WebProtégé both include features to easily check the profile adherence of an ontology under development, simplifying this work. If a task-suitable CODP exists but does not adhere to a required reasoning profile, the author encourages the developer who notices this to reimplement and release that CODP as a new profile-compliant version.

The tradeoff between *Reasoning Performance Efficiency* and *Learnability* is caused by the same underlying mechanism – several OWL constructs that increase learnability of the model are associated with poor reasoning characteristics. This includes the aforementioned use of domain and range restrictions, which typically help users understand the intended use of CODP properties. It also includes the use of class restriction axioms that can similarly help illustrate how classes and properties in a CODP are to be reused, but which in several cases decrease reasoning performance [17]. When constructing the CODPs it may be beneficial to consider increasing the scope of the CODP metadata or documentation to better describe the CODP's features and their intended use, rather than relying on the above discussed performance-affecting constructs for this purpose (unless required to realise some actual functionality of the CODP).

The tradeoff between *Interoperability* versus *Operability* relates to the transitive import closure indicator (*MI25*). For a CODP to be as interoperable as possible, it should be aligned to and share foundational concepts with as many well-established CODPs as possible – which would be indicated by *MI25* having a high value. However, in order for a CODP to be as easy to use as possible, it should not require the developer to have to understand a large number of phenomena that are outside of scope of the problem the CODP intends to solve, i.e., the *MI25* value should be low.

Finally, the tradeoff between *Learnability* versus *Reusability* again concerns indicators *MI17* (property domain restrictions) and *MI18* (property range restrictions). As discussed above, the use of property and range restrictions improve learnability, so a CODP that has higher values for these indicators would be easier to understand and learn than one that has lower ones. However, in order for a CODP to be as reusable as possible it should make as small an ontological commitment as possible while still fulfilling its design goals. Consequently for reusability *MI17* and *MI18* should measure as low as possible, as the properties

defined in the CODP should not make assumptions about what classes they are used together with, but rather be reusable outside of the exact CODP scope if need be. This is particularly important as the semantics of OWL property domain and range definitions imply that if a property has multiple asserted ranges or domains, the resulting inferred domain or range of that property is the intersection of the set of domain or range class expressions. In other words, it is not possible for a child CODP to extend a property domain or range from an imported parent CODP, only to constrain it further.

3.4.3. Improving ODP Tooling

As mentioned in Section 3.1, in order to increase CODP adoption, the research community needs to improve the quality of both CODPs themselves and CODP support tooling and methods. The presented quality indicators provide guidance on what types of features such improved CODP support tooling (including both client-side XD support tooling and online CODP repositories) should include.

The largest and most well-known CODP repository on the web is that hosted by OntologyDesignPatterns.org. The presentation in this repository would benefit from reducing the number of extraneous documentation fields displayed by default for each listed CODP, in order to comply better with indicator *DI04 – Documentation Minimalism* (see Section 3.3.2 for details). In addition, example uses of each CODP should be described, ideally both in text format and with accompanying graphical illustrations (as per indicators *DI05 – Example Illustration Count* and *DI06 – Example Text Count*). While the portal has support for describing usage scenarios, rather few CODPs do so at the time of writing.

There is also room for improvement with regard to more easily finding CODPs compliant with project requirements (particularly in light of the above discussed tradeoffs). Whether users browse the CODP repository manually or use a search engine, they would benefit from being able to constrain or filter the resulting CODP list using the presented indicators (particularly those indicators contributing to degraded reasoning performance, decreased interoperability, or decreased reusability). Related to this, a useful addition to the repository would be if it were to support the notion of different versions or flavours of CODPs, i.e., different solutions or reusable modules to the same problem that vary across these indicators. For instance, there might be a full OWL 2 CODP representing event participation, and a reduced OWL 2 EL version of that same CODP, sacrificing some logic expressiveness for increased reasoning performance.

In addition to the above improvements, the author's ongoing work in several projects indicates that repository usability would be improved by increases in the documentation coverage, by a clearer overview of CODP interdependence, and by harmonisation of the graphical CODP representations used. Thankfully the Association for Ontology Design & Patterns (ODPA), stewards of the portal in question, have recently begun work aiming to improve the quality of the repository, both in terms of features and in terms of content.

Client-side XD support tools and ontology IDE:s would, as touched upon in Section 3.4.1, benefit from better support for constructing and evaluating tests over CODP-based ontologies. For full XD support such test tooling would need be

able to evaluate whether Competency Questions⁵ and Reasoning Requirements of an ontology are fulfilled (for instance by evaluating SPARQL query result over asserted and inferred data respectively), and whether Contextual Statements still hold (possibly using a rules engine, as testing based on open world reasoning would not trigger an error if data expected to be in the model was lacking).

Additionally, such tools would benefit from functions that simplify dealing with CODPs that have large import closures (*MI25 – Transitive Import Count*). For instance, a tool might provide the user with a guided interface that partially clones a reused CODP, keeping only those parts that are required in order to successfully model the solution to the problem at hand, while removing extraneous higher-level classes and properties. Promising work in this direction, the MIREOT guidelines and tooling, has been carried out by Courtot et al.[1] and Hanna et al. [15]. Integrating their work in a CODP and XD workflow has the potential to bring great benefit to CODP users and developers.

3.5. Summary

In this chapter we have presented a CODP Quality Model consisting of a meta-model providing a conceptual understanding and language by which CODP quality can be discussed. We have also presented a set of quality characteristics to consider when using CODPs to develop ontologies, and an initial set of quality indicators by which these quality characteristics can be measured. Some of the tradeoffs that need to be made when prioritising among the presented quality characteristics have been introduced and discussed, and directions for further development (of research and tooling) proposed.

Bibliography

- [1] M. Courtot, F. Gibson, A. L. Lister, J. Malone, D. Schober, R. R. Brinkman, and A. Ruttenberg. Mireot: The minimum information to reference an external ontology term. *Applied Ontology*, 6(1):23–33, 2011.
- [2] M. Doerr, J. Hunter, and C. Lagoze. Towards a Core Ontology for Information Integration. *Journal of Digital Information*, 4(1), 2006.
- [3] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Modelling Ontology Evaluation and Validation. In *The Semantic Web: Research and Applications*, pages 140–154. Springer, 2006.
- [4] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann, R. Gil, F. Bolici, and O. Strignano. Ontology evaluation and validation. Technical report, Laboratory for Applied Ontology, ISTC-CNR, 2005.
- [5] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WordNet with DOLCE. *AI magazine*, 24(3):13, 2003.
- [6] M. Genero, G. Poels, and M. Piattini. Defining and Validating Measures for Conceptual Data Model Quality. In *Advanced Information Systems Engineering*, pages 724–727. Springer, 2006.

⁵For an introduction to CODP requirements, including the types mentioned here, see [24]

- [7] A. Gómez-Pérez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Springer-Verlag, London, Berlin, 2004.
- [8] N. Guarino and C. Welty. Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002.
- [9] N. Guarino and C. A. Welty. An Overview of OntoClean. In *Handbook on Ontologies*, pages 201–220. Springer, 2009.
- [10] K. Hammar. Ontology design patterns in use: lessons learnt from an ontology engineering case. In *Workshop on Ontology Patterns in conjunction with the 11th International Semantic Web Conference 2012 (ISWC 2012)*, 2012.
- [11] K. Hammar. Reasoning performance indicators for ontology design patterns. In *Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns-Volume 1188*, pages 27–38. CEUR-WS. org, 2013.
- [12] K. Hammar. *Towards an Ontology Design Pattern Quality Model*. Linköping University Electronic Press, 2013.
- [13] K. Hammar. Ontology design pattern property specialisation strategies. In *Knowledge Engineering and Knowledge Management*, pages 165–180. Springer, 2014.
- [14] K. Hammar and K. Sandkuhl. The State of Ontology Pattern Research: A Systematic Review of ISWC, ESWC and ASWC 2005–2009. In *Workshop on Ontology Patterns: Papers and Patterns from the ISWC workshop*, pages 5–17, 2010.
- [15] J. Hanna, C. Chen, W. A. Crow, R. Hall, J. Liu, T. Pendurthi, T. Schmidt, S. F. Jennings, M. Brochhausen, and W. Hogan. Simplifying mireot: a mireot protégé plugin. In *11th International Semantic Web Conference ISWC 2012*, page 25. Citeseer, 2012.
- [16] ISO. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, International Organization for Standardization, 2011.
- [17] Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy. Predicting Reasoning Performance Using Ontology Metrics. In *The Semantic Web – ISWC 2012*, pages 198–214, 2012.
- [18] O. I. Lindland, G. Sindre, and A. Solvberg. Understanding Quality in Conceptual Modeling. *Software, IEEE*, 11(2):42–49, 1994.
- [19] S. Lodhi and Z. Ahmed. Content Ontology Design Pattern Presentation. Master’s thesis, Jönköping University, 2011.
- [20] S. Lohmann, S. Negru, F. Haag, and T. Ertl. VOWL 2: User-oriented visualization of ontologies. In *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW ’14)*, volume 8876 of *LNAI*, pages 266–281. Springer, 2014.
- [21] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 2(15):1–18, 2004.
- [22] D. L. Moody and G. G. Shanks. What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In *Entity-Relationship Approach – ER ’94 Business Modelling and Re-Engineering*, pages 94–111. Springer, 1994.

- [23] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. *Software Engineering, IEEE Transactions on*, 28(6):595–606, 2002.
- [24] V. Presutti, E. Blomqvist, E. Daga, and A. Gangemi. Pattern-based ontology design. In *Ontology Engineering in a Networked World*, pages 35–64. Springer, 2012.
- [25] T. L. Saaty. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology*, 15(3):234–281, 1977.
- [26] C. Welty and N. Guarino. Supporting ontological analysis of taxonomic relationships. *Data & Knowledge Engineering*, 39(1):51–74, 2001.

This page intentionally left blank

Chapter 4

On the Roles of Logical Axiomatizations for Ontologies

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

In this brief chapter, we will elaborate on the different roles which logical axiomatizations can play for ontology design patterns and for ontologies in general. While doing this, we also encounter some of the many open research questions regarding this issue.

4.1. Logic Choices

Logic-based knowledge representation has a long-standing history [4] and can, e.g., be traced back to Aristotle's *Organon*. As part of Computer Science, it has been a mainstay of the Artificial Intelligence field since its inception in the 1950s.

The resulting breadth and depth of research, results, and applications led to the adoption of logic-based knowledge representation as the foundation for information representation languages on the World Wide Web [6], and thus for the Semantic Web, including the W3C standards RDF [18], RIF [9], and OWL [5].

These standards, of course, capture only the essence of logic-based knowledge representation, which could be (and sometimes is) used for knowledge representation on the Web. In fact, the discussions are ongoing on suitable formalisms or modifications and extensions of the existing standards [7]. Let us briefly mention some of the key dimensions of these investigations.

Monotonic versus mon-monotonic logics, open versus closed world assumption. First a clarification: Monotonicity (meaning, additional information cannot cause the withdrawal of previously derived logical consequences) is mostly associated with the open world assumption, while non-monotonicity is mostly associated with the closed world assumption. The distinction is not quite as clear a cut upon a closer look though [20].

Description Logics [17], and thus the Web Ontology Language OWL [5] are based on the *open world assumption* (and are monotonic). Intuitively this means that *absent* information is viewed as *unknown*, e.g., a listing of the eight planets of the solar system does not preclude that there may be any additional ones out there [2]. This choice seems very sensible from a *Web* perspective where additional information may be as easy to obtain as by crawling a few more Web pages.

However, some situations call for an invocation of the *closed world assumption*, the most obvious perhaps being that of database access, where the information in the database is to be considered *complete*. E.g., if a person is not listed as booked on a particular flight by the airline's internal information system, then it should be assumed that the person is in fact *not* booked on this flight. Some of these cases can in fact already be captured by the use of *nominals* as provided by description logics and OWL; and this essentially constitutes a type of world closure captured within a monotonic logic.

More complex cases arise when non-monotonicity is required. Such cases arise, e.g., when the abovementioned database access scenario is coupled with deductive inferences, in the sense that a class shall consist of exactly those instances for which it can be derived deductively that they belong to that class. Assume, for example, that a knowledge base allows us to derive that both David and Raghava are graduate students, but that it does not allow us to derive any information about the graduate student status of Cogan. Under the closed world assumption, we then would this consider a proof that Cogan is in fact *not* a graduate student. If, however, we later add additional information, say that Cogan is a graduate student, then the previous conclusion of Cogan not being a graduate student would have to be retracted. Thus, reasoning under this logic would be non-monotonic.

The question of monotonicity or non-monotonicity of a reasoning task can often be used as a first check whether the task can be expressed in OWL. E.g., in [21] it was investigated whether a relation *more biodiverse than* between regions can be expressed in OWL, if this relative biodiversity is being assessed by means of the number of different species occurring in these regions. The task is non-monotonic since our assessment may change when further species are found. The example also highlights that non-monotonicity may arise out of some type of *introspection*, e.g. from reflecting on what is or is not known based on the given data.

While many non-monotonic logics have been defined in the past four decades, three of the initial proposals have arguably been the most prolific: McCarthy's Circumscription [22] which focuses on world closure as key intuition, Moore's Autoepistemic Logic [23] which focuses on introspection, and Reiter's Default Logic [24] which captures reasoning of the type *every bird flies, unless we have evidence for it not to fly*.

In the context of Semantic Web and ontology modeling, non-monotonic issues often arise naturally, as in the biodiversity case or the database case discussed above. While such non-monotonicity cannot be captured by the OWL standard, research investigations are ongoing regarding the possible extension of OWL, or more general description logics, by non-monotonic features. Such extended knowledge representation languages, which combine open and closed world aspects, are often referred to as *local closed world* languages. A significant body of work has been done on such formalisms; rather than repeating earlier enumerations, we simply refer to [10] which contains a relatively recent overview.

Schema-centric versus data-centric approaches. Similar to the above discussed distinction between open and closed world logics, the distinction between schema-centric and data-centric knowledge representation and reasoning approaches is not entirely clear cut. Contrasting them, however still helps in clarifying different emphases of different logics.

On the schema-centric side of the spectrum, e.g., are traditional description logics such as *ALC* [6] without ABoxes, in which it is not possible to talk about instances (or constants), but which still allow reasoning over class expressions. On the other hand, rule-based approaches such as logic programming [19] and its variants do not natively support reasoning over predicate relationships, but focus on the derivation of facts involving instances (i.e., constants).

This contrast between description logics and, broadly speaking, *rules* has for most of a decade defined a major dividing line (and point of contention) within the Semantic Web field, however more recently this gap is closing [1, 14–16].

Also of relevance in this context is the use of *integrity constraints* in the context of ontologies: While the semantics of OWL axioms is *inferential*, in the sense that it makes it possible to derive (or infer) additional knowledge from the given axioms, integrity constraints constitute constraints which the data has to satisfy in order to be compatible with a schema. E.g., such a constraint could specify that for any person with a bank account a physical address must be on file. Integrity constraints are usually non-monotonic, in the sense that absence of such an address in the example just given would cause an inconsistency, while later addition of an address would reinstate consistency, which cannot happen in a monotonic setting. Mature proposals for adding integrity constraints to OWL have been made [26]. Some aspects of the forthcoming RDF Shapes Constraint Language (SHACL) [11] may also address integrity constraint issues.

The importance of standardization and tools. A central dividing line between different logics for ontologies on the Web is that some such logics have been standardized, and some have not. Some, such as RDF, RIF and OWL mentioned above, have further been standardized by the World Wide Web Consortium (W3C) for explicit use in a Semantic Web context. As a consequence, such standardized languages have gained significant visibility, and in its wake also a landscape of diverse compatible tools, which range from research prototypes to commercial solutions.

Hence, currently, the Web Ontology Language OWL is the primary language of choice for expressing ontologies. This does not necessarily mean, of course,

that it is the best possible conceivable language for this purpose, or that it will not undergo major revisions and extensions in the future. In particular, and as already mentioned above, some things – such as non-monotonic constructs, integrity constraints, some type of rules [13], and other constructs of potential importance for ontology modeling [12] – are not expressible in OWL.

4.2. Axioms for Inferential Reasoning

Arguably, the most obvious use of logical axioms is for inferential reasoning, i.e. to derive logical consequences from information contained in the ontology or knowledge base. Typical examples for schema-centric reasoning would be the derivation of inferred class relationships, e.g., if *Feline* is a subclass of *Mammal*, which in turn is a subclass of *Animal*, then it can be inferred that *Feline* is a subclass of *Animal*. Corresponding data-centric reasoning may infer, e.g., that a *Feline* identified as *Mimi* is also an *Animal*.

The formal semantics of standards such as OWL and RDF in fact is tailored towards inferential reasoning, and significant research efforts are made towards developing ever more efficient algorithms and systems for performing inferential reasoning [3].

Strong inferential reasoning systems for ontologies are of course often tailored towards the standardized representation languages, although exceptions exist, such as the support, by many systems, of the Semantic Web Rule Language SWRL [8], which has only the status of a W3C Member Submission.

4.3. Axioms to Inform Humans

Another, often undervalued use of logical axioms in ontology modeling is that of informing humans about the intended meaning or classes and their relationships. Axioms stated for the purpose of inferential reasoning can of course already help humans to understand the intended meaning of ontology parts, and we therefore would argue that such axioms, in human-readable form, should be an important part of the documentation of an ontology, and not only be published as part of an OWL file.

In the practice of ontology modeling, in fact, we sometimes encounter axioms which quite clearly are not intended for inferential reasoning, because they constitute tautologies. However, they are informative for humans as to the intended meaning of terms and their relations. A particular case in point would be minimum cardinality statements with the minimum specified as zero, e.g. *A person always has at least 0 children* or *A chess game has at least 0 moves*. Such statements clarify that properties such as *hasChild* or *hasHalfMove* are intended for use together with the classes *Person* and *ChessGame*, respectively, which for example could indicate to a human user that they are not to be used in the context of trees (as data structures) or the game of Go (where it would be more appropriate to use *ply* or *turn*).

Informing humans, and thereby disambiguating the meaning of ontology terms, is also a primary reason why it is sometimes useful to state axioms in

documentation which cannot be expressed in the standardized language chosen to formally represent the ontology. This would be the case, e.g. for axioms which can be expressed in first-order predicate logic, but not in description logic fragment which constitutes OWL DL.

And sometimes the axioms governing a part of an ontology can be expressed in OWL, but the result would be rather difficult to read and thus assess by a human. E.g., the example statement *All elephants are bigger than all mice* from [25] can easily be expressed using a (first-order) rule such as

$$\text{Elephant}(x) \wedge \text{Mouse}(y) \rightarrow \text{biggerThan}(x, y),$$

but the corresponding formulation in the description logic corresponding to OWL DL would require the introduction of two new properties, and a total of three axioms:

$$\begin{aligned} \text{Elephant} &\equiv \exists R_1.\text{Self} \\ \text{Mouse} &\equiv \exists R_2.\text{Self} \\ R_1 \circ R_2 &\sqsubseteq \text{biggerThan} \end{aligned}$$

While the OWL version can of course be used in the corresponding OWL file, for the documentation the rule variant may be preferable, since its meaning is more immediately obvious.

4.4. Axioms as Integrity Constraints

An inferential axiom would state that *every person has a birth place*, while a corresponding integrity constraint would state that *every person has a birth place listed* (in this knowledge base). In the first case, if a birth place for a person, say *John*, is not listed or inferable from the data, then this would not constitute an inconsistency: According to the open world assumption, John would still have a birth place, it would just be the case that we don't know about it (yet). In the case of the integrity constraint, absence of a birth place for John would cause an inconsistency (or other exception).

Using OWL (in description logic syntax), the inferential axiom could be expressed, e.g., as

$$\text{Person} \sqsubseteq \exists \text{hasBirthPlace}.\text{Place}.$$

For the integrity constraint version, we know of no standardized ontology language to express it. However, it would be conceivable to, e.g., abuse OWL syntax and use the very same axiom with a disclaimer (for the human reader) that it is to be read as an integrity constraint. This would serve the purpose of informing humans about the intended meaning. The axiom could still be used for inferential semantics of course, it would simply not derive anything if the integrity constraint is satisfied. And a human user interested in checking the integrity constraint would have the option of using other mechanisms for this purpose, e.g., by retrieving all members of the class Person and all triples involving the hasBirthPlace property, and to then check whether each person has a birth place listed.

4.5. Axioms as Shape Constraints

Shape constraints in the context of OWL have not been studied prominently yet. Only recently, they have been taken up by a W3C working group to standardise a shape constraint language for RDF [11]. However, they have often made informal appearances in the form of class diagrams and tools which attempt to create class diagrams from OWL files.

We do not yet sufficiently understand the role of shape constraints for ontologies. Anecdotal experience – from ontologies we have looked at – indicates that domain and range declarations for properties may sometimes be intended as a type of shape constraints, informing the human user about the class diagram in a non-visual form. This is of course in contrast to the standardized semantics, which is inferential also for domain and range declarations. Likewise, tautological axioms such as the minium zero cardinality axioms discussed above, can also play the informal role of shape constraints.

The formalization of shape constraints for OWL ontologies will certainly lead to a non-monotonic formalism, which also indicates that shape constraints cannot be expressed in the monotonic Web Ontology Language.

4.6. Conclusion

We have briefly discussed different major approaches to logical knowledge representation as they pertain to theory and practice of ontology modeling and ontology language research. We have also provided four mostly distinct perspectives on the roles which logical axioms can play in order to disambiguate the meaning of ontology constructs or for programmatic use.

We are not aware of any systematic investigations into the different roles axioms do, can, or could play for ontology modeling and practice. However we believe that it should be a worthwhile endeavour to embark on such investigations.

Acknowledgements. This work was partially supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*.

Bibliography

- [1] A. Amarilli and M. Benedikt. Combining existential rules and description logics. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2691–2697. AAAI Press, 2015.
- [2] K. Batygin and M. E. Brown. Evidence for a distant giant planet in the Solar system. *The Astronomical Journal*, 151(2):22, 2016.

- [3] M. Dumontier, B. Glimm, R. S. Gonçalves, M. Horridge, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, G. B. Stamou, and G. Stoilos, editors. *Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015) co-located with the 28th International Workshop on Description Logics (DL 2015), Athens, Greece, June 6, 2015*, volume 1387 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [4] D. M. Gabbay and J. Woods, editors. *The Handbook of the History of Logic*. Elsevier, Amsterdam, 2004–2014. 11 Volumes as of April 2016.
- [5] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at <http://www.w3.org/TR/owl2-primer/>.
- [6] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2010.
- [7] P. Hitzler, J. Lehmann, and A. Polleres. Logics for the semantic web. In D. M. Gabbay, J. Woods, and J. Siekmann, editors, *Logic and Computation*, volume 9 of *Handbook of the History of Logic*, pages 679–710. Elsevier, Amsterdam, 2014.
- [8] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission 21 May 2004, 2004. Available from <http://www.w3.org/Submission/SWRL/>.
- [9] M. Kifer and H. Boley, editors. *RIF Overview*. W3C Working Group Note 22 June 2010, 2010. Available from <http://www.w3.org/TR/rif-overview/>.
- [10] M. Knorr, P. Hitzler, and F. Maier. Reconciling OWL and non-monotonic rules for the Semantic Web. In L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. Lucas, editors, *ECAI 2012, 20th European Conference on Artificial Intelligence, 27-31 August 2012, Montpellier, France*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 474–479. IOS Press, Amsterdam, 2012.
- [11] H. Knublauch and D. Kontokostas, editors. *Shapes Constraint Language (SHACL)*. W3C Editor’s Draft 10 April 2016, 2016. Available from <http://w3c.github.io/data-shapes/shacl/>.
- [12] A. Krisnadhi, P. Hitzler, and K. Janowicz. On capabilities and limitations of OWL regarding typecasting and ontology design pattern views. In V. Tamma, M. Dragoni, R. Goncalves, and A. Lawrynowicz, editors, *Ontology Engineering. 12th International Experiences and Directions Workshop on OWL, OWLED 2015, co-located with ISWC 2015, Bethlehem, PA, USA, October 9-10, 2015, Revised Selected Papers*, volume 9557 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 2016. To appear.
- [13] A. Krisnadhi, F. Maier, and P. Hitzler. OWL and Rules. In A. Polleres et al., editors, *Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 382–415. Springer, Heidelberg, 2011.
- [14] M. Krötzsch. *Description Logic Rules*, volume 008 of *Studies on the Semantic Web*. IOS Press/AKA, 2010.
- [15] M. Krötzsch, F. Maier, A. A. Krisnadhi, and P. Hitzler. A better uncle for

- OWL: Nominal schemas for integrating rules and ontologies. In S. Sadagopan, K. Ramamritham, A. Kumar, M. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th International World Wide Web Conference, WWW2011, Hyderabad, India, March/April 2011*, pages 645–654. ACM, New York, 2011.
- [16] M. Krötzsch, S. Rudolph, and P. H. Schmitt. A closer look at the semantic relationship between datalog and description logics. *Semantic Web*, 6(1):63–79, 2015.
 - [17] M. Krötzsch, F. Simančík, and I. Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.
 - [18] O. Lassila and R. R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation 10 February 2004, 2004. Available from <http://www.w3.org/TR/REC-rdf-syntax/>.
 - [19] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.
 - [20] D. Makinson. *Bridges from Classical to Nonmonotonic Logic*, volume 5 of *Texts in Computing*. King’s College Publications, 2005.
 - [21] D. C. Martínez, K. Janowicz, and P. Hitzler. A logical geo-ontology design pattern for quantifying over types. In I. F. Cruz, C. A. Knoblock, P. Kröger, E. Tanin, and P. Widmayer, editors, *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL’12, Redondo Beach, CA, USA, November 7-9, 2012*, pages 239–248. ACM, 2012.
 - [22] J. McCarthy. Circumscription – A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13(1–2):27–39, 1980.
 - [23] R. Moore. Possible-worlds Semantics for Autoepistemic Logic. In *Proceedings of the 1984 Non-monotonic Reasoning Workshop*. AAAI, Menlo Park, CA, 1984.
 - [24] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.
 - [25] S. Rudolph, M. Krötzsch, and P. Hitzler. All Elephants are Bigger than All Mice. In F. Baader, C. Lutz, and B. Motik, editors, *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*, 2008.
 - [26] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

Chapter 5

Dolce+D&S Ultralite and its main ontology design patterns

Valentina Presutti, STLab, ISTC-CNR, Italy

Aldo Gangemi, LIPN, Université Paris 13, CNRS UMR7030, and ISTC-CNR, Italy

5.1. Introduction

This chapter aims at providing a hand note on the Dolce+D&S UltraliteUltralite ontology (DUL)¹ and its main ontology design patterns (ODP).

ODPs are modelling solutions to recurrent modelling problems [19]. Following this emerging idea, introduced in [5, 16], the OntologyDesignPatterns.org² (odps.org) initiative started in 2008, initially developed within a european research project³, and then evolved thanks to a community effort. The web portal is currently maintained by the Association of Ontology Design and Patterns (ODPA). It hosts a catalogue of ODPs, the first of which were derived from DUL. The design process of DUL, in fact, was characterised by a selection of modelling components from complex theories, and their integration. This made DUL a perfect source for extracting general-domain ODP at the time of the kickoff of the ODP catalogue.

It is important to note that the extracted ODPs are independent from DUL's commitment, by which we mean that the overall ontological commitment of DUL, as a whole, is not inherited by the extracted ODPs. For example, DUL describes the world by distinguishing between objects and events, but one may want to use the *Information Realisation* ODP (cf. [18] and Section 5.3.10) in an ontology that does not commit to this distinction. This is possible, as the *Information Realisation* ODP does not take any explicit position on such a distinction. Similarly, the same ODP can be used in an ontology, such as DUL, that does commit

¹<http://www.ontologydesignpatterns.org/ont/DUL/dul.owl>

²<http://www.ontologydesignpatterns.org>

³Lifecycle of Networked Ontologies (NeOn) EU FP7 Integrated Project, 2006-2010

to this distinction. In other words, *ODPs are only committed to the description of the small portion of the world that they encode*. From the moment that these ODPs were extracted from DUL, they have been isolated, that is, each of them is an independent ontology with the function to provide a modelling solution to a specific recurrent modelling problem.

Another important note is the distinction between an ODP and its implementations. For example, odps.org provides links to one or more OWL files, for many of the catalogued ODPs. One may decide to directly reuse such implementations by importing them in a local ontology. Alternatively, the ODPs can be directly implemented locally (i.e. used as a template) and possibly aligned with other existing implementations, based on the same ODP. Both these solutions realise an ODP-based ontology design approach. The only issue to be considered nowadays is that there is lack of tools for automatically recognising ODPs within ontologies, hence reflecting their presence by means of implementation strategies, e.g. by separating namespaces or implementing them in separate files, may help in ontology alignment and ODP reuse. However, this is only a technological limit that can be overcome by research efforts in the near future. A helpful tool for partially addressing this lack is to document an ontology with the ODPs it reuses and implements. A UML profile for component diagrams, aimed at supporting this task, is introduced in this chapter and its use is demonstrated by means of some examples.

The chapter is structured as follows: Section 5.2 provides a description of DUL and of the main theories that it is based on. Section 5.3 gives an overview of the basic ODPs extracted from DUL by listing them and providing an informal description of their modelling issues and proposed solutions. Section 5.4 introduces a UML profile for component diagrams that supports the design of ODP-based ontologies and describes how DUL can be design as a composition of ODPs. Finally, Section 5.5 provides a note about projects and research work that have reused DUL, as a foundational ontology.

5.2. Dolce+D&S Ultralite: DOLCE for the Semantic Web

In this section, we give a brief overview of the original formal theories that are at the base of the Dolce+D&S Ultralite ontology (DUL), which in turn is the original source of a set of domain-independent ontology design patterns (ODPs) that constituted the initial seeds for the ontologydesignpatterns.org (odps.org) catalogue. DUL is a lightweight version of an ontology that combines five formal theories: the “Descriptive Ontology for Linguistic and Cognitive Engineering” (DOLCE) [27], its main component, the “Description and Situation” (D&S) ontology [17], the Plan ontology [29], the ontology of Information Objects (IO) [4], and the Collection ontology [8].

5.2.1. DOLCE: Main ontological principles and commitments

DOLCE is a foundational ontology. Its original design was made in the S5 modal logic [11]. DOLCE covers general distinctions concerning physical and social objects, events, abstractions, attributes, dimensional regions, as well as mereological

(part), participation, inherence (attributive), and localisation relations. In other words, DOLCE covers some of the core design modelling issues that are commonly met in the majority of ontology projects.

DOLCE categories are thought of as cognitive artefacts, ultimately depending on human perception, ontological commitment is not constrained to quantify only what *really exists*, i.e. that can be scientifically proved.

5.2.1.1. Endurants and perdurants

The fundamental DOLCE assumption and commitment is the distinction between *enduring* and *perdurating* entities. This difference is related to their behaviour with respect to time. *Endurants* are present, during their existence, at any time, together with all their proper parts, i.e. they are *wholly* present. For example, a music track is wholly present at each instant of its existence. The presence of *perdurants* instead is only partial: they are not fully present at multiple instants, but are made of temporal parts, e.g. phases. For example, a temporal part of the *act of listening to a music track* at time t_1 is not present at a successive time t_2 .

A consequence is that endurants can change over time as whole entities, i.e. the same entity can have different incompatible characteristics at different times, while perdurants cannot exhibit this behaviour as their temporal parts do not keep the same identity over time. Compare Coltrane's Giant Steps track from a 1960 vinyl record, and its playing on your turntable: the first is wholly there at any time, and can get scratched at some time because of its frequent laying, but the playing of the Giant Steps track is never wholly there at any time, and the possible scratches will change the playing event, making it a different experience for the listener.

The main relation between endurants and perdurants is that of *participation*: an endurant, e.g. a person, participates in a perdurant, e.g. a meeting. More intuitively, the distinction between endurants and perdurants can be traced to the difference between objects of any kind, and events.

Endurant and Perdurant are two of the four disjoint top level classes of DOLCE. The other two top classes are *Quality* and *Abstract*.

5.2.1.2. Qualities

Qualities in DOLCE refer to the *inherent* aspects of entities. The shape, color, length, etc. of an entity are its qualities. However, it is important to not confuse a quality with its value, which is called *quale* (plural: *qualia*): a quality exists jointly with its entity, and two entities never have the same quality, although their respective different qualities can have the same associated value, in their reference value space. For example, “the color of Paris’ sky at the sunset” and the “color of somebody’s eyes” are different qualities, each constantly dependent on its respective entity. Nevertheless, they may have the same value (i.e. position) in the color space. Hence, each quality *type* has an associated quality *space* with a specific structure. For example, lengths are usually associated with a metric linear space. Regions (e.g. a particular shade of red) within quality spaces (e.g. the color space) identify values for specific qualities. The relation between an entity and its qualities is named *inherence*, i.e. a quality inheres to an entity.

Notice that space and time locations in DOLCE are special types of qualities, whose values belong to the spatial and temporal regions, respectively. Qualities can also have qualities themselves, for example the color of somebody's eyes can have a specific luminosity.

5.2.1.3. Abstracts

Quality regions, quality spaces, and qualia (specific values) are *abstract* entities. Their main characteristic is that they are not localised either in space or time, and are disjoint from qualities.

5.2.1.4. Main relations

Parthood: DOLCE defines two types of parthood relations: atemporal and time-indexed. The former holds between perdurants or abstracts, i.e. a keynote speech is part of a conference, while the latter holds for endurants, e.g. my computer has a new hard-disk since two months. Parthood is not defined for qualities.

Constitution: expresses the relation that for example exists between a statue and the clay it is made of. Constitution is defined for perdurants and for endurants. DOLCE follows the philosophical position that considers constitution as different from *identity*. In fact, the constitution relation requires co-localisation (i.e. located in the same space) and co-presence (i.e. existence at the same time) of its arguments to hold, but the persistence conditions of them is different: e.g. the statue can survive a change of one of its part, while the clay cannot. Also their histories differ, e.g. a clay existed before the statue did.

Participation: DOLCE defines a time-indexed participation relation that holds between endurants and perdurants, i.e. endurants participate in perdurants, at a certain time.

Inherence: this relation holds between entities and their qualities, i.e. the color of Paris' sky at sunset *inheres* to Paris.

Quale: DOLCE also has a *quale* relation, which holds between a quality and its value (an instance of Quale from a quality region), in an associated quality space. The quale relation can be time-indexed or not based on the nature of the entity (object, event) the quality inheres to.

Figure 5.1 summarises the main concepts and relations of DOLCE, also showing what categories can be related by what relations.

5.2.2. The Description and Situation ontology

The ontology of “Descriptions and Situations” D&S, originally encoded in KIF [20], is also a foundational ontology, and defines a vocabulary for roles, frames, concepts, and situations, which represents many domains (e.g., biomedicine, law, business, organisations) that are often ambiguous in using words for expressing either individuals, concepts, or relations. For example, the term *Person* can be conceptualised as either a class of persons, or a juridical role of an entity based on legal norms, or even as a relation between a legal norm and a specific person. OWL2 provides the *punning* mechanism to make a same term denote different and separate interpretations, but does not give any hint on how to use punning

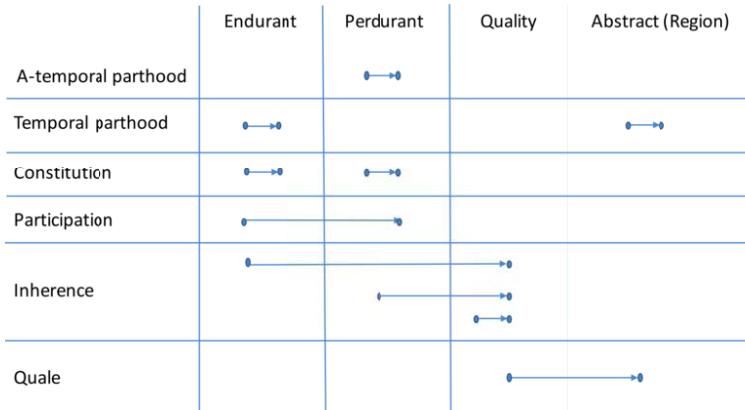


Figure 5.1. Main concepts and relations of DOLCE. Columns indicate categories (which are all disjoint with each other), while rows indicate relations. An arrow connecting two cell means that the corresponding relation is legal between the two corresponding categories, a missing arrow means that the corresponding relation is illegal between the two corresponding categories.

to represent typical use cases such as legal norms vs. cases, medical diagnoses vs. physiological events, plan models vs. plan executions, etc. D&S provides that framework by distinguishing *Descriptions* (reified intensional relations) vs. *Situations* (reified extensional relations); *Concepts* (reified intensional classes) vs. *Entities* (any entity playing a role in a situation). Relations are provided as well: a description possibly *defines* a concept, a situation possibly *satisfies* a description, a concept possibly *classifies* an entity.

5.2.3. The Plan ontology

The Plan Ontology specializes D&S to describe plan models and their executions, task, role and parameter concepts and their flow, goals, etc. The basic classes *Plan*, *Plan Execution*, *Goal*, *Task*, *Role*, *Parameter* are introduced and reused by DUL.

5.2.4. The Information Object ontology

Information entities include all social objects that are created and/or used by cognitive systems to communicate, reason, and stipulate new entities in a cognitive or social world. The original distinction between information objects and their realisations lies in the ability to materialise, manifest, and replicate information across different media, based on which for example the copyright of the content of a published work is different from the copyright of the material book, of a specific copy of that book, of a lecture using that content, etc. A thorough analysis of these concepts was presented in [4], which defines the Information Object ontology (IO). [18] presents a set of patterns, including the ones extracted from DUL, that are useful in any use case that includes information entities as first-order entities, i.e. not just as labels or comments.

5.2.5. The Collection ontology

Collections and collectives can be conceptualised as first-order entities in many use cases, distinguishing them from abstract sets of entities. For example clubs, teams, exhibitions, and constellations are all objects that are perceived as unique only by virtue of a shared property across its members, which does not make them instances of the whole collection: a Monet's painting is not an instance of the Musée Marmottan collection (in fact it is not a collection), it is a member of it. Collection and collectives have been deeply studied and formalised in [8], which is one of the input theories of DUL.

5.2.6. DOLCE+D&S Ultralite

The previously described theories, i.e. DOLCE, D&S, Plan, Information Object, and Collection, are at the basis of the DOLCE+D&S Ultralite⁴ (DUL) ontology. DUL is the result of various refinements and integrations of the OWL versions of those theories. In particular, the interested reader may want to inspect DOLCE-Lite⁵, and compare it with DUL. DOLCE-Lite was designed in order to maximally approximate the semantics of the original DOLCE in OWL, and to reflect its terminological choices. In other words, DOLCE-Lite is the result of a philological translation of DOLCE into OWL, only limited by OWL expressivity⁶. Unfortunately, DOLCE-Lite was hardly usable, its terminology far from being friendly for the average Semantic Web user. These are the main motivations why DUL was conceived: (i) intuitive terminology, (ii) lighter axiomatisation, (iii) integration of other theories.

DUL reduces the axiomatisation to the essentials needed to reflect the main principles and commitments of DOLCE, such as e.g. disjointness between the top level categories and main relations with their expected domains and ranges. In lieu of using the terms *endurant* and *perdurant*, it introduces the more intuitive *object* and *event*, respectively. But this is not the only introduced terminological change, also relation names are more intuitive, e.g. *hasQuality* instead of *inheres*.

DUL also integrates D&S in DOLCE, e.g. introducing the concept of Situation, which provides a generic vocabulary for supporting the representation of *n*-ary relations, hence enabling the representation of time-indexed relations such as participation, and parthood. It is not straightforward to integrate the constructive perspective of D&S with the commitments taken by DOLCE. In fact, D&S provides an extremely flexible way of describing the world, as it leaves the observer free to express the most appropriate commitments according to a specific modelling context and its functional goals. Considering that DOLCE is a theory, we can represent it as an instance of the D&S *Description* concept. At the same time, *Descriptions* and *Situations* results from constructive definitions, i.e. they derive from someone's observation. Hence, in DOLCE, they are classified as

⁴<http://www.ontologydesignpatterns.org/ont/DUL/dul.owl>

⁵<http://www.ontologydesignpatterns.org/ont/dlp/DOLCE-Lite.owl>

⁶Notice that temporal- and possible worlds- indexing of DOLCE relations were left out from DOLCE-Lite.

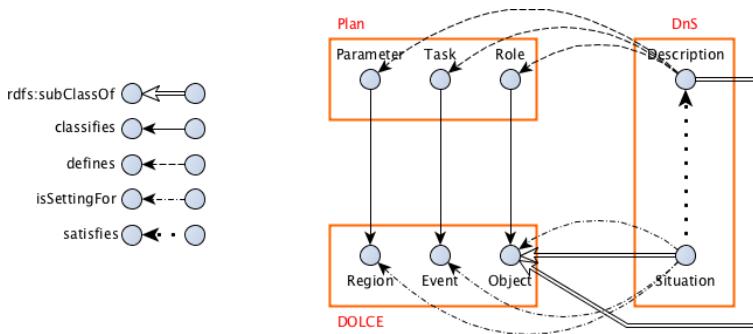


Figure 5.2. Main concepts and relations of DOLCE+D&S Ultralite.

special types of *Social objects*, which is a sub class of *DUL Objects*, i.e. DOLCE endurants.

The main concepts of the Plan ontology enrich the overall conceptual architecture of DUL. These are *Role*, *Task* and *Parameter*, which instances can be used for classifying those of *Object*, *Event*, and *Region*, respectively.

Figure 5.2 shows a schematic view of the conceptual architecture of DUL, by highlighting the different theoretical components from which it generates.

Finally, DUL integrates concepts from the Information Object (IO) and Collection ontologies. From IO, the classes *Information Object* and *Information Realization* are reused, and the relation *realizes* holding between them. From Collection, the classes *Collection* and *Collectives* (for human groups and societies) and the relation *has member* are reused.

5.3. The main ontology design patterns from DOLCE+D&S Ultralite

This section lists the main ODPs extracted from DUL and each of them is individually and independently described in an informal way, by providing a description of the modelling issue that it can be a solution for, and of how this solution is implemented.

5.3.1. Events and participation

Modelling issue: I want to describe my world by distinguishing between objects (that exist over time) and events (that happen in time), and to represent the fact that objects are participating entities in events.

Modelling solution: Two disjoint classes, i.e. *Object* and *Event*; an object property *participates in*, whose domain is *Object* and its range is *Event*. The inverse object property *has participant*. An existential axiom, which states that objects participate in some event (an object occurs in at least one *participates in* triple), and another for expressing that events have at least one participant.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/participation.owl>.

5.3.2. Quality of entities

Modelling issue: I want to represent that entities have unique qualities (including qualities themselves), which may be associated with values identified in a quality space.

Modelling solution: A class *Quality*, representing entity qualities, which is the range (domain) of an object property *has quality* (inverse, *is quality of*) for associating entities to their specific qualities. A class *Quality Space* disjoint with *Quality*, which represents the sum of the abstract regions that provide values to qualities. An object property *has region* (and its inverse *is region for*), which domain (range) is *Quality* and its range (domain) is *Quality Space*. An existential axiom, which assesses that qualities have at least one quality space (a quality occurs in at least one *has region* triple), and vice versa, a quality space is associated with at least one quality. Quality spaces are also named regions. This ODP may be conveniently combined with *Region* (see next section).

5.3.3. Region

This ODP is often combined with the *Sequence* ODP Section 5.3.9) and the *Part-whole* ODP (c.f. Section 5.3.7).

Modelling issue: I want to represent abstract dimensional spaces as first order entities, which can be used as values for qualities of entities. For example, the color space and the distance space.

Modelling solution: A class *Region*, which represents abstract dimensional spaces. A datatype property *has region data value*, with domain *Region*, used for encoding values associated to a region. An object property *has region* (and its inverse *is region for*) used for associating entities to regions. Usually, this ODP is combined with the *Quality of entities* ODP (see Section 5.3.2). Other ODPS that are commonly combined with this ODP are *Part-whole* (c.f. Section 5.3.7) and *Sequence* (c.f. Section 5.3.9).

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/region.owl>.

5.3.3.1. Time Interval

This ODP can be modelled as a specialisation of the *Region* ODP (c.f. Section 5.3.3).

Modelling issue: I want to represent time intervals as first order entities in my domain, and express facts about them in my knowledge base. For example, my vacation week is the same as my husband's.

Modelling solution: A class *Time Interval*, which represents time spaces. A datatype property *has interval date* that encodes typed values for a time interval; a same time interval can have more than one value: begin date, end date, date at which the interval holds, as well as dates expressed in different formats. Object properties, analogous to those defined in the ODP *Region* (c.f. Section 5.3.3) should be defined in order to allow the association of entities to time intervals, e.g. *has time interval*.

Sample implementation:

[`http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl`](http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl)⁷

5.3.4. Entity Classification

This ODP (and its specialisations) is commonly used in combination with the *Description* and *Situation* ODPS (c.f. Section 5.3.11 and 5.3.12, respectively) or their specialisations (e.g. *Plan*, c.f. Section 5.3.11.1).

Modelling issue: I want to represent concepts that I use for classifying entities of my domain and I want to express facts on these concepts too in my knowledge base. For example, I want to classify my documents based on the main topics that they are about, and I want to associate such topics to statistical information.

Modelling solution: A class *Concept*, which instances are used for classifying things. This classification relation is represented by an object property *classifies* (inverse, *classified by*) that has *Concept* as its domain (range).

Sample implementation:

[`http://www.ontologydesignpatterns.org/cp/owl/classification.owl`](http://www.ontologydesignpatterns.org/cp/owl/classification.owl).

5.3.4.1. Role Classification

This ODP can be modelled as a specialisation of the *Entity Classification* ODP (c.f. Section 5.3.4)

Modelling issue: I want to represent roles that things play as first-order entities of my domain and I want to express facts on these roles in my knowledge base. For example, I want to represent the fact that a person plays the role of project manager and I want to associate each role to a minimum fee.

Modelling solution: A class *Role*, which instances are used for classifying things according to the roles they play. This relation is represented by an object property *is role of* (inverse, *has role*) that has *Role* as its domain (range).

Sample implementations:

[`http://www.ontologydesignpatterns.org/cp/owl/objectrole.owl`](http://www.ontologydesignpatterns.org/cp/owl/objectrole.owl)

[`http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl`](http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl)

5.3.4.2. Parameter Classification

This ODP can be modelled as a specialisation of the *Entity Classification* ODP (c.f. Section 5.3.4). It is commonly combined with the *Region* ODP (c.f. Section 5.3.3).

Modelling issue: I want to represent constraints (or selection) on observable values. For example, I want to define the concept “very high” and associate it to a certain region in a value space, e.g. the region identifying height for buildings.

Modelling solution: A class *Parameter* and a class *Region*. Parameters are used for classifying regions. This classification relation is represented by an object property *is parameter of* (inverse, *has parameter*, that has *Parameter* as its domain (range).

⁷The pattern is fully compatible to W3C OWL Time and other time ontologies.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/parameter.owl>.

Notice that the same modelling issue can be addressed with a different pattern, that shortcuts regions by modelling them with datatype properties. In this case we would only define the class *Parameter* and a datatype property *is parameter for region data value*, having *Parameter* as domain and the appropriate datatype as range.

5.3.4.3. Task Classification

This ODP can be modelled as a specialisation of the *Entity Classification* ODP (c.f. Section 5.3.4).

Modelling issue: I want to classify actions according to the task that they allow to accomplish. For example, the action of buying a flight ticket to a certain location, may be classified by the task of organising a working trip. Commonly, tasks are defined in the context of a plan or workflow, but this ODP only considers the specific classification relation between a task and an action.

Modelling solution: A class *Task* and a class *Action*. Tasks are used for classifying actions. This classification relation is represented by an object property *is task of* (inverse, *has task*, that has *Task* as its domain (range)). If it is appropriate, the object property's label may be less general, for example *task addressed by* (inverse, *addresses task*).

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/taskexecution.owl>

5.3.5. Collection and Membership

This ODP is commonly combined with the *Entity classification* ODP, which allows to express common properties of a collection's members, which characterise the collection.

Modelling issue: I want to represent containers for entities that share one or more common properties, and have them as first-class entities in my knowledge base. For example, “the Louvre Egyptian collection”.

Modelling solution: A class *Collection* that represent containers and that is the domain (range) of an object property *has member* (inverse, *is member of*), which represents the relation between a collection and its members.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/collectionentity.owl>.

Notice: one may expect that an existential axiom stating that a collection must have at least one member is defined. However, there are cases in which we may want to talk about collections, without knowing what members they have/had. Consider for example the historical and archeological domains, where you may want to have in your knowledge base an entity representing a collection that is known or supposed to exist in the past (e.g. Aristotle's exotic works), without knowing exactly what was its content.

5.3.6. Location

Modelling issue: I want to represent location of entities in a very generic sense. For example, a political geographic entity (Roma, Lesotho), a non-material location determined by the presence of other entities (“the area close to Roma”), pivot events or signs (“the area where the helicopter fell”), complements of other entities (“the area under the table”).

Modelling solution: A class *Place* that represents an approximate location for things. Precise locations can be modelled with the *Region* ODP (c.f. Section 5.3.3). An object property *is location of* (inverse, *has location*), whose domain (range) is *Place*. An existential axiom stating that a Place is location for at least one thing.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/place.owl>.

5.3.7. Part-whole

Modelling issue: I want to represent that things can have parts, and derive that something is part of a thing because it is a part of its part. For example, the engine is part of the car, the piston is part of the engine, hence the piston is part of the car.

Modelling solution: A transitive object property *has part* (inverse, *is part of*).

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/partof.owl>.

5.3.7.1. Component

This ODP can be modelled as a specialisation of the *Part-whole* ODP (c.f. Section 5.3.7).

Modelling issue: I want to represent a non transitive part-whole relation. For example, if the following facts hold: (i) the engine is part of the car, and (ii) the piston is part of the engine, then I cannot conclude that the piston is part of the car. This ODP is useful to model direct parts of things that show a design that structure them.

Modelling solution: An object property *has component* (inverse, *is component of*).

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/componency.owl>.

5.3.8. Constitution

Modelling issue: I want to represent the layering structure of things, where intuitively a constituent is a part of something belonging to a lower layer. For example, scientific granularities (e.g. body-organ-tissue-cell) or ontological “strata” (e.g. social-mental-biological-physical) are typical layerings. Since layering is actually a partition of the world described by the ontology, constituents are not properly classified as parts. A desirable advantage of this distinction is that we

are able to talk e.g. of physical constituents of non-physical objects (e.g. systems), while this is not possible in terms of parts. In all these examples, we notice a typical discontinuity between the constituted and the constituent object: e.g. a social system is conceptualised at a different layer from the persons that constitute it, a person is conceptualised at a different layer from the molecules that constitute them, and a river is conceptualised at a different layer from the atoms that constitute it.

Modelling solution: an object property *has constituent* (inverse, *is constituent of*). Local universal restrictions may be defined according to the layering of the world described by the ontology.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/constituency.owl>.

5.3.9. Sequence

Modelling issue: I want to represent a sequence schema between entities, including the case in which they overlap. For example, “year 1999 precedes 2000”, “deciding what coffee to use” precedes “preparing coffee”, “World War II follows World War I”, “in the Milan to Rome autoroute, Bologna precedes Florence”.

Modelling solution: an transitive object property *precedes* (inverse, *follows*) for expressing the sequence relation between entities. A symmetric object property *overlaps* for representing the case in which entities overlap.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/sequence.owl>.

5.3.10. Information Realisation

Modelling issue: I want to represent information objects, e.g. musical compositions, texts, pictures, and their concrete realisations, e.g. a written document containing the text of a law, a musical track containing a musical composition, as distinct objects in my knowledge base.

Modelling solution: Two disjoint classes *Information Object* and *Information Realisation* and an object property *is realised by* (inverse, *realises*) that relates an information object to its concrete realisations. An existential axiom stating that an information object is realised by at least one information realisation, as well as a dual existential axiom stating that an information realisation realises at least one information object.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/informationrealization.owl>.

5.3.11. Description

This ODP is commonly used in combination with the *Situation* and the *Entity classification* ODPs (c.f. Section 5.3.12 and 5.3.4, respectively).

Modelling issue: I want to represent conceptualisations as first-order entities in my knowledge base. For example, a workflow is a description of some actions

(tasks) to be executed by some objects (roles) complying to certain constraints (parameters). In this sense a description defines a number of concepts. For example, the “law on same-sex marriage” is a description that defines the concepts “partner” and “parenting”.

Modelling solution: A class *Description*, which represents conceptualisations, and a disjoint class *Concept*, which represents the concepts that are defined within a description. An object property *defines* (inverse, *is defined by*) that represents the relation between descriptions and their defined concepts.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/description.owl>.

5.3.11.1. Plan

This ODP can be modelled as a specialisation of the *Description* ODP (c.f. Section 5.3.11) It is commonly combined with the *Entity classification* ODP and its specialisation and with the *Sequence* ODP (c.f. Sections 5.3.4 and 5.3.9, respectively).

Modelling issue: I want to represent plans, their goal and their structure. For example, “making american coffee” is a plan having a goal of “drink a coffee” that defines the tasks “pour water in the reservoir”, “put the carafe on the heating plate”, and “add the desired amount of grounds to the filter”.

Modelling solution: A class *Plan* with a relation, represented by an object property *defines* (inverse, *is defined by*), to a (disjoint) class *Concept*. Then combine it with the *Entity classification* ODP (c.f. Section 5.3.4) and its specialisations, i.e. the *Parameter classification*, the *Role classification* and the *Task classification* ODPs. A class *Goal*, which represents the plan’s goal. Then integrate the *Component* ODP (c.f. Section 5.3.7.1) and add a local existential restriction on the property *has component*, stating that a plan must have at least one component of type goal.

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/basicplan.owl>.

5.3.12. Situation

This ODP is commonly used in combination with the *Description* ODP (c.f. Section 5.3.11) and its specialisations. In fact, by combining these two ODPs, the core D&S ODP is obtained.

Modelling issue: I want to represent observed states of affair together with their relational context. For example, the actions that have been taken in the execution of a plan. More in general, I want to represent *n*-ary relations.

Modelling solution: A class *Situation* and an object property *is setting for* between a situation and the things that are involved in it.

Sample implementations:

<http://www.ontologydesignpatterns.org/cp/owl/situation.owl>

<http://www.ontologydesignpatterns.org/cp/owl/basicplanexecution.owl>

5.3.13. Time-indexed participation

This ODP is not extracted from DUL, however can be modelled as a specialisation of the *Situation* ODP. Including it here is meant to show how the *Situation* ODP can be used as a general reference, and specialised for representing n -ary relations, for example time- and space-indexed relations. This ODP is often used in combination with the *Time Interval* and the *Sequence* ODPs (c.f. Section 5.3.3.1 and Section 5.3.9, respectively).

Modelling issue: I want to represent participating situations that hold for a certain time period. For example, “the suspect was at the party from 9pm to 11pm”.

Modelling solution: A class *Time Indexed Participation*, which represents the participation situation. An object property *has participant* (inverse *is participant in*), which represents the relation between the participation situation and the things that participate in it. An object property *at time*, the domain of which is *Time Indexed Situation* and the range of which is a class *Time Interval*, disjoint with *Time Indexed Participation* that represents periods of time, and that can be implemented with the *Time Interval* ODP (c.f. Section 5.3.3.1).

Sample implementation:

<http://www.ontologydesignpatterns.org/cp/owl/timeindexedparticipation.owl>.

Notice that the same modelling issue can be addressed with a different pattern, that shortcuts time intervals by modelling them with datatype properties. With this solution, the object property *at time* is replaced by a datatype property *at time*, having *Time Indexed Participation* as domain and the appropriate datatype as range.

5.4. A UML-based notation for module-based ontologies.

In order to show a ODP-based view of Dolce+D&S Ultralite, a UML profile for component diagrams is introduced in this section. A profile is an extension mechanism to customise UML models for specific domains or platforms. Component diagrams are originally meant to depict how components are wired together to form larger components and/or software systems.

The UML profile here introduced extends the Ontology Definition Metamodel (ODM) [30] (which provides stereotypes for both class diagrams and packages) by defining stereotypes for component diagrams, which enable the representation of ODPs as components that implement and/or reuse certain interfaces.

The concept of ontology interface has been defined in the literature related to ontology modularisation: in [12] the authors propose the notions of *import interface* and *export interface*. A different notion of *ontology interface* is investigated in [15], whose main feature is to support knowledge encapsulation. The idea is that an ontology module defines its content by means of *interfaces* and these interfaces constitute the access point to its knowledge base.

The ODM profile extension is depicted in Figure 5.3. Two stereotypes are defined that can be used within UML components diagrams: *Ontology Module* (OM) and *Ontology Design Pattern* (ODP). The latter inherits from the former



Figure 5.3. Ontology Module and ODP stereotypes for UML Component Diagrams

and has a tagged value *intent*, i.e., a multi-line text that describes the modelling problem addressed by the ODP.

Figure 5.4 shows how this notation can be used for depicting an ontology as a composition of modules and ODPs. The “Sample Module” component identifies an ontology module that *realise realises*, i.e. implements, a concept “M” (which can be implemented as a class or a property), while it *uses* a concept “C”, which is realised by the “Sample ODP C”. If we make a distinction between the design and the implementation time of an ontology project, a *uses* interface may represent, at design time, concepts used by a module that are planned to be needed but whose modelling is delegated to other modules, which will be possibly identified at a later stage. For this reason a diagram drawn at design time may not reflect the actual vocabulary used by the ontology modules that will be implemented.

At implementation time, a *uses* interface represents concepts used by a module that are realised by another identified module (such as in Figure 5.4). In this case, the realising module exposes interfaces that reflect the actual vocabulary used in its implementation.

The sample ontology depicted in Figure 5.4 also shows that modules can be related by a specialisation relation, such as the case of “Sample ODP B” that specialises “Sample ODP A”. ODPs are special modules, designed in order to solve a specific and focused modelling problem. This is captured, in this notation, by a tagged value named “*intent*”, the content of which is shown in the diagram as an (optional) note attached to the ODP.

The actual design and implementation of a module (or ODP) may be represented with a class diagram, hence it is important to notice that component diagrams are not meant to show axiomatisation details. Their goal is to show the main concepts captured by an ODP and to give the intuition of the modelling issue they solve. For this reason, component diagrams can also be a good documentation means for sharing modelling choices with domain experts.

5.4.1. Composing ODPs

The current implementation of DUL is constituted by a monolithic OWL ontology, although its conception followed an ODP-based design approach, as it is evidenced by the ODPs that could be extracted from it. Using the notation introduced in the previous section, Figure 5.5 shows DUL as the composition of its main ODPs.

This diagram is meant to demonstrate how an ontology can be designed and illustrated as a composition of ODPs. It is worth noting that the use of ODPs does not always pair with the reuse of some specific implementation of them, nor does ODP use have to be reflected in the organisation of ontology namespaces or files: an ontology can be a composition of ODPs even though it is implemented as a monolithic OWL file, whose vocabulary terms are defined in a same namespace.

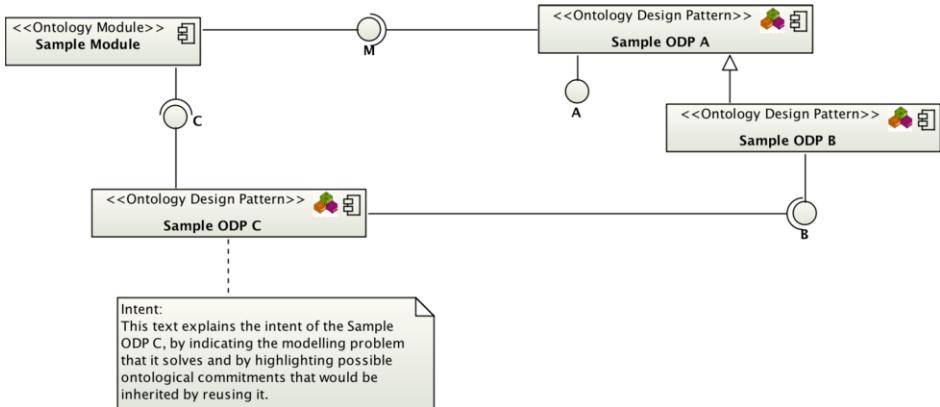


Figure 5.4. A sample diagram using the Ontology Module and ODP stereotypes, for depicting an ontology as a composition of ontology modules and ODPs.

In other words, the reuse of ODPs is an ontology design choice, not constrained by a specific organisation of the ontology implementation.

However, at the moment, there is lack of methods to automatically recognise ODPs within ontologies; hence ODP-based modularisation is a pragmatic choice enabling ODP use with currently available tools. The UML profile used in this section can be helpful for documenting reused ODPs and the way they are combined, at both design and implementation phases of an ontology project, and for both ontologies that reflect or not such modularised design in their implementation strategies.

Figure 5.5 also helps showing how the composition of ODPs allows us to express ontological commitments that are not proper of individual ODPs. As mentioned earlier in Section 5.3, ODPs are independent on each other and they commit only to the partial view of the world that they encode. Still they can be used for designing an ontology that makes a strong and specific commitment on the distinctions to be adopted in the description of the world, such as DUL does. The axioms defined for composing them as well as the ones used for specialising them allow to achieve this result.

Let us assume that DUL did not exist before the ODPs described in Section 5.3. We want to obtain this ontology by composing ODPs. We want to distinguish between objects and events, where the former participate in the latter. Hence, we reuse the *Events and participation* ODP (c.f. Section 5.3.1). Events and objects are top classes in DUL, together with abstract regions and qualities. Hence, we also include the *Region* (and its specialising ODPs) and the *Quality of entities* ODPs (c.f. Section 5.3.3 and Section 5.3.2, respectively). The three ODPs constitute the basis of DUL. In order to formally represent DUL ontological commitment, a disjointness axiom among the classes defined by these three ODPs, i.e. *Object*, *Event*, *Region*, and *Quality*, is added. Furthermore, as it is depicted in the diagram, the *Quality of entities* and the *Region* ODP are connected by their respective interfaces. The same applies to the *Events and participation*

and the *Quality of entities* ODPs.

5.4.2. Designing DOLCE+D&S Ultralite as a composition of ontology patterns

As we said, DUL also wants to support the core *Description and Situation* ODP (c.f. Figure 5.2), hence in order to model it, *Description*, *Situation*, and the *Entity classification* ODPs are combined. The specialising ODPs of *Entity classification* are also added. In order to comply with DUL ontological commitment, all classes defined in these ODPs are defined as subclasses of (social) *Object*⁸. As the diagram shows, the three ODPs are combined by their respective interfaces. In addition to the basic modelling of the three patterns, the *Situation* ODP is extended in order to realise the interface *satisfies*, *is satisfied by* that is needed by the *Description* ODP, in order to model the relation between descriptions and situations. This interface is implemented by means of two object properties (inverse of each other) that allow to model that a description may be satisfied by some situation and vice versa, that situations may satisfy some description. Alternatively, this additional interface could have been realised as an extension of the *Description* ODP or as specific to a module, which includes the three ODPs, as shown in Figure 5.6.

The *Information Object*, *Collection and membership* and *Location* ODPs are also added to the ontology. Again, in order to comply with DUL ontological commitment, the classes that these ODPs define are integrated in the ontology as subclasses of social objects. The ODPs modelling the relevant relations for DUL are also included, i.e. *Part-whole*, *Sequence*, and *Constitution*.

Appropriate axioms are added in order to compose these ODPs with the previous ones. For example, a local universal axiom is added for the *has part* object property when its domain is the class *Concept*, defined in the *Entity classification* ODP, stating that concepts have only concepts as their parts. Similarly, where appropriate, other local restrictions are added.

5.5. Notes about Dolce+D&S Ultralite reuse

DUL has been reused and applied in many domains and contexts; in this section, a non exhaustive list is reported.

DUL is extended by a domain ontology used by an e-learning system for prosthetic schools [7]. For e-learning systems, it has been also used in [25]. DUL is at the base of the hydro ontology [10], which has the aim of fostering interoperability among datasets collecting water data. Several works in the domain of smart products and multimedia in general reuse DUL: for describing smart product objects [23], for modelling multimedia annotation facets [35], for multimedia content annotation [1], and for audiovisual formal descriptions [22].

DUL has been used as foundational ontology in the legal domain in many projects, we report here the work presented in [13], which describes an ontology

⁸The class *Object* is further specialised by two disjoint classes *Social Object* and *Physical Object*. Descriptions, situations and concepts are social objects, according to DUL.

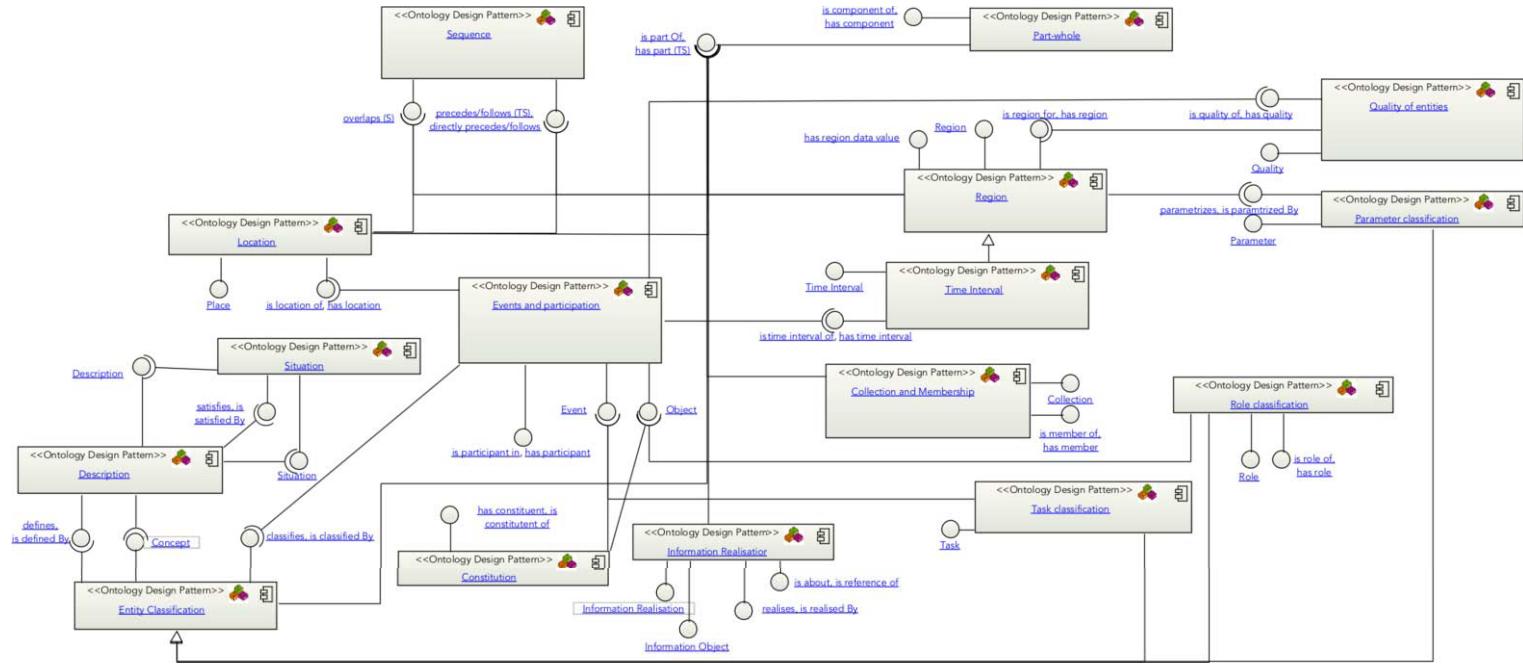


Figure 5.5. DOLCE+D&S Ultralite refactored as a composition of Ontology Design Patterns.

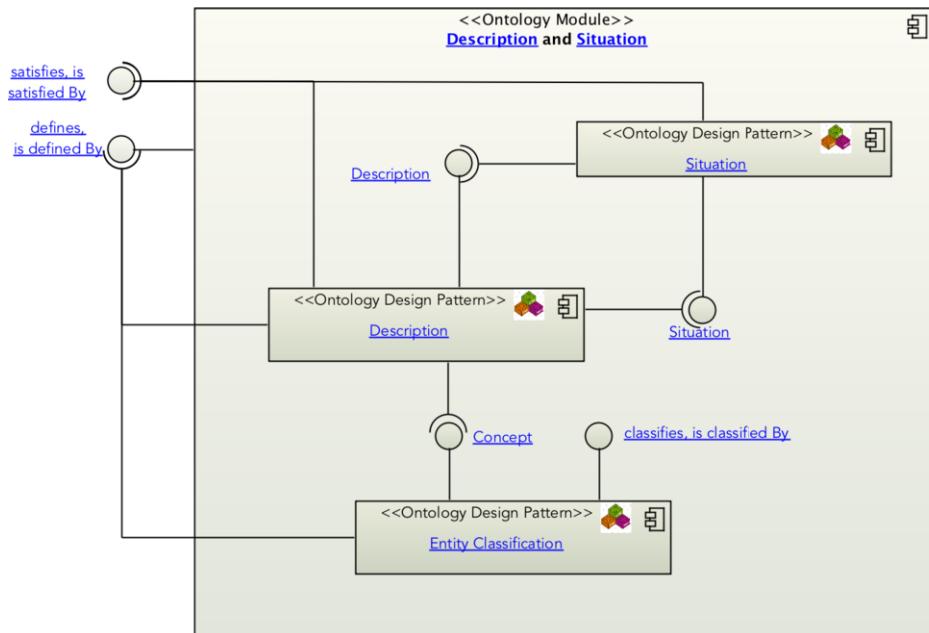


Figure 5.6. The *Description* and *Situation* module designed as a composition of the *Description*, *Situation* and *Entity classification* ontology design patterns.

constructed from european legislative text. For modelling events, it has been used in [38]. For managing geo-spatial data, DUL has been reused in [3, 14, 37].

In the medical domain, DUL is at the base of the development of several ontologies: for modelling intracranial aneurysms [6], for annotating medical images [39] and neuroimages [21], and for modelling the biomedical research domain [36].

Other domains in which DUL has been reused as foundational ontology are: robotics and automation [34], textile manufacturing (the AsISKnown project)⁹, food transformation and wine making [28], cybersecurity [31], enterprise integration [9], process mining [33], disaster management [2], semantic sensor network [24], and customer relationship management (CRM) [26].

Finally, DUL has been recently used for identifying and fixing inconsistencies in the DBpedia ontology [32].

5.6. Conclusion

In this chapter, an overview of the Dolce+D&S Ultralite (DUL) and its main ontology design patterns (ODP), was given. After a description of the main theories and principles that are at the basis of its foundation, we have listed the main ontology design patterns that could be extracted from DUL and isolated as

⁹<http://www.asisknown.org/>

independent solutions to recurrent modelling problems. Each ODP is presented by providing a description of the addressed modelling issue, a description of the modelling solution that it encodes, and a reference to a sample OWL implementation of it. We then look at DUL from a different perspective, as if it did not exist before the described ODPs. After introducing a notation for drawing ODP-based ontology diagrams (a UML-profile for component diagrams extending the Ontology Definition Metamodel) [30], we discuss how ODPs can be composed for modelling ontologies, and show how DUL can be (re-)designed an ODP-based ontology. Finally, a short note reporting some of the research work and projects that have reused DUL, is presented.

Bibliography

- [1] T. Athanasiadis, V. Tzouvaras, K. Petridis, F. Precioso, Y. Avrithis, and Y. Kompatsiaris. Using a multimedia ontology infrastructure for semantic annotation of multimedia content. In *Proceedings of the 5th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2005) at the 4th International Semantic Web Conference (ISWC 2005)*, 2005.
- [2] G. Babitski, F. Probst, J. Hoffmann, and D. Oberle. Ontology design for information integration in disaster management. In *Informatik 2009: Im Focus das Leben, Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck, Deutschland, Proceedings*, pages 3120–3134, 2009.
- [3] J. Bateman and S. Farrar. Towards a generic foundation for spatial ontology. In *Formal ontology in information systems: proceedings of the third conference (FOIS-2004)*, page 237, 2004.
- [4] W. Behrendt, A. Gangemi, W. Maass, and R. Westenthaler. Towards an ontology-based distributed architecture for paid content. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 257–271, 2005.
- [5] E. Blomqvist and K. Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In C. C. et al., editor, *ICEIS Proc. of the 7th Int. Conf. on Enterprise Information System*, pages 413–416, 2005.
- [6] M. Boeker, H. Stenzhorn, K. Kumpf, P. Bijlenga, S. Schulz, and S. Hanser. The@ neurist ontology of intracranial aneurysms: providing terminological services for an integrated it infrastructure. In *AMIA*, 2007.
- [7] C. M. Bogdan. Domain ontology of the virdent system. *International Journal of Computers Communications & Control*, 6(1):45–52, 2011.
- [8] E. Bottazzi, C. Catenacci, A. Gangemi, and J. Lehmann. From collective intentionality to intentional collectives: An ontological perspective. *Cognitive Systems Research*, 7(2-3):192–208, 2006.
- [9] A. Bouras, P. Gouvas, and G. Mentzas. ENIO: an enterprise application integration ontology. In *18th International Workshop on Database and Expert Systems Applications (DEXA 2007), 3-7 September 2007, Regensburg, Germany*, pages 419–423, 2007. DOI:10.1109/DEXA.2007.25.

- [10] B. Brodaric and T. Hahmann. Towards a foundational hydro ontology for water data interoperability. In *In Proceedings of the 11th Int. Conference on Hydroinformatics (HIC-2014)*, 2014.
- [11] B. F. Chellas. *Modal logic: an introduction*, volume 316. Cambridge Univ Press, 1980.
- [12] M. d'Aquin, P. Haase, S. Rudolph, J. Euzenat, A. Zimmermann, M. Dzbor, M. Iglesias, Y. Jacques, C. Caracciolo, C. B. Aranda, and J. M. Goméz. D1.1.3 NeOn Formalisms for Modularization: Syntax, Semantics, Algebra, 2008.
- [13] S. Despres and S. Szulman. Construction of a legal ontology from a european community legislative text. In *Jurix*, pages 79–88, 2004.
- [14] S. Duce. Towards an ontology for reef islands. In *GeoSpatial Semantics*, pages 175–187. Springer, 2009.
- [15] F. Ensan and W. Du. A knowledge encapsulation approach to ontology modularization. *Knowledge and Information Systems*, 26(2):249–283, 2010. DOI:10.1007/s10115-009-0279-y.
- [16] A. Gangemi. Ontology design patterns for semantic web content. In Y. G. et al., editor, *Proc. of the 4th Int. Semantic Web Conference, ISWC 2005*, volume 3729 of *LNCS*, pages 262–276. Springer, 2005.
- [17] A. Gangemi and P. Mika. Understanding the semantic web through descriptions and situations. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, pages 689–706, 2003. DOI:10.1007/978-3-540-39964-3_44.
- [18] A. Gangemi and S. Peroni. The information realization pattern. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. In this volume, chapter 15. IOS Press/AKA Verlag, 2016.
- [19] A. Gangemi and V. Presutti. Ontology Design Patterns. In S. Staab and R. Studer, editors, *Handbook on Ontologies, 2nd Edition*, International Handbooks on Information Systems, pages 221–243. Springer, Berlin, Germany, 2009. DOI:10.1007/978-3-540-92673-3_10.
- [20] M. R. Genesereth and R. E. Fikes. Knowledge interchange format-version 3.0: reference manual. Technical report, Computer Science Department, Stanford University Stanford, California, USA, 1992.
- [21] B. Gibaud, G. Kassel, M. Dojat, B. Batrancourt, F. Michel, A. Gaignard, and J. Montagnat. NeuroLOG: sharing neuroimaging data using an ontology-based federated approach. *Annual Symposium proceedings [electronic resource]. AMIA Symposium*, 2011:472–80, 2011.
- [22] A. Isaac and R. Troncy. Designing and using an audio-visual description core ontology. In *Workshop on Core Ontologies in Ontology Engineering*, pages 5–8, 2004.
- [23] S. Janzen and W. Maass. Smart product description object (spdo). In *Poster Proceedings of the 5th International Conference on Formal Ontology in Information Systems (FOIS)*, Saarbrücken, Germany, 2008.
- [24] L. Lefort, C. Henson, K. Taylor, P. Barnaghi, M. Compton, O. Corcho, R. Garcia-Castro, J. Graybeal, A. Herzog, K. Janowicz, et al. Semantic

- sensor network xg final report. Technical report, W3C Incubator Group Report, 2011.
- [25] L. Lemnitzer, E. Mossel, K. Simov, P. Osenova, and P. Monachesi. Using a domain-ontology and semantic search in an e-learning environment. In M. Iskander, editor, *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, pages 279–284. Springer Netherlands, Dordrecht, 2008. DOI:10.1007/978-1-4020-8739-4_49.
 - [26] D. Magro and A. Goy. Towards a first ontology for customer relationship management. In *Proceedings of the 5th International Conference on Soft Computing As Transdisciplinary Science and Technology*, CSTST '08, pages 637–643, Cergy-Pontoise, France, 2008. ACM. DOI:10.1145/1456223.1456352.
 - [27] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology Library (final), 2003.
 - [28] A. Muljarto, J. Salmon, P. Neveu, B. Charnomordic, and P. Buche. Ontology-based model for food transformation processes - application to winemaking. In *Metadata and Semantics Research - 8th Research Conference, MTSR 2014, Karlsruhe, Germany, November 27-29, 2014. Proceedings*, pages 329–343, 2014. DOI:10.1007/978-3-319-13674-5_30.
 - [29] D. Oberle, S. Lamparter, S. Grimm, D. Vrandecic, S. Staab, and A. Gangemi. Towards ontologies for formalizing modularization and communication in large software systems. *Applied Ontology*, 1(2):163–202, 2006.
 - [30] Ontology definition metamodel, September 2014.
 - [31] A. Oltramari, L. F. Cranor, R. J. Walls, and P. D. McDaniel. Building an ontology of cyber security. In *Proceedings of the Ninth Conference on Semantic Technology for Intelligence, Defense, and Security, Fairfax VA, USA, November 18-21, 2014.*, pages 54–61, 2014.
 - [32] H. Paulheim and A. Gangemi. Serving dbpedia with DOLCE - more than just adding a cherry on top. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, pages 180–196, 2015. DOI:10.1007/978-3-319-25007-6_11.
 - [33] C. Pedrinaci and J. Domingue. Towards an ontology for process monitoring and mining. In *CEUR Workshop Proceedings*, volume 251, pages 76–87, 2007.
 - [34] E. Prestes, J. L. Carbonera, S. R. Fiorini, V. A. M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. J. S. Gonçalves, M. E. Barreto, M. K. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.
 - [35] M. Romanelli, P. Buitelaar, and M. Sintek. Modeling linguistic facets of multimedia content for semantic annotation. In B. Falcidieno, M. Spagnuolo, Y. Avrithis, I. Kompatsiaris, and P. Buitelaar, editors, *Semantic Multimedia: Second International Conference on Semantic and Digital Media Technologies, SAMT 2007*, pages 240–251. Springer Berlin Heidelberg, Genoa, Italy, 2007. DOI:10.1007/978-3-540-77051-0_26.
 - [36] M. Samwald and K. Adlassnig. The bio-zen plus ontology. *Applied Ontology*, 3(4):213–217, 2008. DOI:10.3233/AO-2008-0053.

- [37] S. Schade. *Ontology-driven translation of geospatial data*. PhD thesis, University of Münster, 2010.
- [38] A. Scherp, T. Franz, C. Saathoff, and S. Staab. F-a model of events based on the foundational ontology dolce+ dns ultralight. In *Proceedings of the fifth international conference on Knowledge capture*, pages 137–144. ACM, 2009.
- [39] L. Temal, M. Dojat, G. Kassel, and B. Gibaud. Towards an ontology for sharing medical images and regions of interest in neuroimaging. *Journal of Biomedical Informatics*, 41(5):766–778, 2008. DOI: 10.1016/j.jbi.2008.03.002.

This page intentionally left blank

Chapter 6

Multi-layered n-ary Patterns

Aldo Gangemi, LIPN, Université Paris 13 and ISTC-CNR

Valentina Presutti, STLab, ISTC-CNR

6.1. Problem addressed

The research area called *knowledge pattern science* [17], tries to discover, represent, reuse, extract, and reason with structures called *knowledge patterns*, which emerge as *relevant invariances* from either structured or unstructured content: ontologies, schemas, databases, web links, social graphs, natural language, etc. In the vein of Christopher Alexander's work on design patterns [1], we call *relevant invariances* the structures that are key for solving problems, and therefore are closely associated to requirements, centrality in graphs and linguistic parse trees, user-orientation in interaction, etc.

In this chapter, we show an example of knowledge pattern science by empirically studying solutions for representing extensional and intensional¹ *multigrade predicates*, widely known as *n-ary relationships* in web ontologies and linked data. A multigrade predicate (cf. [40]) is a predicate having a polymorphic arity, where each *argument place* can have multiple *positions*. Any place or position can be mandatory or optional, leading to polymorphism. Neo-Davidsonian [8] event types are a well known example of multigrade predicates. For example, *preparing a coffee* may involve a preparer, some coffee mix, a machine, some water, a final product, making it at a certain time and place, following certain steps, performing certain actions, etc. While the denoted event always involves most of the mentioned entities, the representation of that event tends to take into account only some of them, e.g. the preparer and the final product, or also indexing it with the time of the preparation. The inability to gather a full representation of the world, and the consequent variability in the requirements of an ontology or data, leads to big interoperability problems when multiple databases or knowledge graphs are integrated.

¹The difference is between actually occurring *relationships*, and *relation* types.

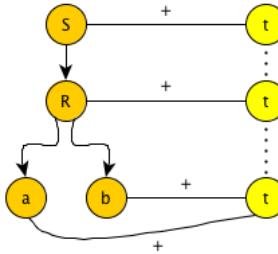


Figure 6.1. The parse tree of Hayes' time-indexing solutions, with the “trickling-down” time parameter.

N-ary relations are everywhere in human interaction. They underlie natural language interpretation (cf. conceptual frames), relational database tables, when recognizing or talking about events, when interpreting news, when making medical diagnoses, interpreting legal norms, controlling situations, or simply preparing a coffee. Most knowledge representation problems require more than two entities to be bound together, and cannot be easily solved using only unary and binary predicates.

The problem is not only technical, i.e. how to represent an n-ary relation with limited expressivity languages, since it also involves design and usability issues, let alone conceptualization problems. The last ones arise for example in many problem-solving tasks that require reasoning over alternative conceptualizations of the same situation or set of facts. Section 6.6.1 below includes typical cases when such reasoning is needed.

This chapter presents a short summary of multidisciplinary literature on representing n-ary relations, then proceeds by presenting the technical and design (dis)advantages deriving from alternative ontology design patterns, firstly for extensional multigrade predicates, called here *situations*, and secondly addressing solutions for the representation of intensional multigrade predicates, called here *descriptions* or (*conceptual*) *frames*.

6.2. Literature summary

The amount of multidisciplinary work done for the representation of multigrade predicates pairs the relevance of the real world use cases involved. Here we provide a simple annotated bibliography, leaving to the curious reader the chance to deepen one or more of the mentioned problems.

There is a long history of solutions to deal with relations that range over more than two arguments, or have polymorphic arities. (Generic) use cases and solution proposals come from AI (e.g. *situation calculus* [31], *relation reification* [10, 11, 22, 29], Minsky's *frames* [33], *description logics* [2, 9, 35], etc.); from philosophy (e.g. *Gestalt theory* [27], Davidson's *reified events* [8], and the debate on the nature of events vs. propositions [34]); from linguistics (e.g. Bach's *eventualities* [3], *Discourse Representation Theory* [25], FrameNet *frames* [4, 39],

cognitive linguistics *schemata* [12]); from conceptual modeling (e.g. [19]); and from the Semantic Web: *n-ary relation reification* [13, 14, 37], *fluents* [5, 45], *named graphs* [7], *linked data integration* [23], *RDF data structure extensions* [28], *RDF reification* [21], *usability* [43]), etc.

Based on the literature, we have singled out the main patterns that are used for multigrade predicate representation at either the extensional or intensional layers.

6.3. Solutions

6.3.1. Three logical patterns to time indexing of sentences

Three general logical patterns (which we call Hayes' patterns because of the author of the reference paper) to express extensional multigrade predicates consist in moving additional argument places within an axiom. An excellent summary was provided by [20] with respect to temporal indexing (i.e. adding a temporal argument place) to sentences of the form $R(a, b)$:

- I. *Attach it to the sentence*, meaning that the sentence $R(a, b)$ is true at the time t . This pattern is called 3D.
- II. *Attach it to the relation (a “fluent”) as an extra argument*: $R(a, b, t)$ This pattern is called 3D+1.
- III. *Attach it to the object terms*, (using a suitable function, written here as an infix @): $R(a@t, b@t)$, where the @ operation means the t -slice of the thing named. This pattern is called 4D.

Hayes [20] shows that the different patterns are syntactic variants, which (in principle) could be unified by an algorithm, which works on a parse tree representation of the three solutions, where the time indexing parameter “trickles down” from 3D to 4D, i.e. from sentences to relational constants to individual constants. Given the three propositions 6.1,6.2,6.3,

$$(R(a,b))+t \quad [sentence\ indexing] \quad (6.1)$$

$$R(a,b) + t \quad [relation\ indexing] \quad (6.2)$$

$$R(a+t, b+t) \quad [individual\ indexing] \quad (6.3)$$

the parse tree (Fig. 6.1) shows the trickling down of the time parameter from the sentence level to the individual level through the relation level. The unification suggested by Hayes however works well with a formal language that has enough expressivity to encode the topology of the parse tree. For example, this is straightforward in first-order logic with some meta-level sugar² (propositions 6.4,6.5,6.6), but not in a description logic.

$$\text{holdsAt}(R(a,b)), t \quad (6.4)$$

$$R(a,b,t) \quad (6.5)$$

$$R(h(a,t), h(b,t)) \quad (6.6)$$

²Hayes adds a sort for time-indexed terms of the vocabulary, so requiring a sorted FOL.

Notice that in FOL we need some appropriate design add-ons to represent the solutions effectively: the `holdsAt` predicate is meta-level sugar for a proposition holding at a certain time, while the dyadic function `h` (standing for *history*) encodes the “time slices” of the individuals, e.g. the time slice of `a` at `t` is the history `h(a, t)`.

But besides those add-ons, FOL design preserves the basic topology of the relation, i.e. a sub-tree remains invariant across transformations, as Figure 6.1 shows: the parse tree from statements to relations to individuals is invariant in the three solutions. Moreover, the different designs do not affect the constants of the vocabulary, which are the same `{R, a, b, t}` (the `holdsAt` relational constant and `h` are meta-level sugar).

On the contrary, when attempting to represent those solutions in formal languages with restricted expressivity, we do usually affect relation topology (e.g. because of relation reification in OWL, which changes the parse tree) and/or vocabulary constants (e.g. because of individual time slice reification in OWL, which adds new individual constants). This means that for languages with restricted expressivity we need to understand the design consequences of each solution, which can eventually impose representation and reasoning constraints.

6.4. Seven RDF-OWL patterns for *n-ary* relation representation

In [18], we have investigated the representation of the three Hayes’ logical patterns in RDF and OWL knowledge bases, and we have singled out seven RDF and OWL2 logical patterns. Then we have compared the resulting models against a multi-dimensional design space, including the following dimensions:

- (a) amount of axioms needed for the representation of each design pattern, calculated on a normalized set of 10000 individuals per ontology
- (b) expressivity of the resulting model, in terms of description logic varieties
- (c) time needed to check the consistency of the model
- (d) time needed to classify the model
- (e) amount of newly generated constants needed
- (f) ability to support DL reasoning with reference to the full n-ary model (cf. Sect. 6.3.1)
- (g) ability to support polymorphism (possibility to add new arguments when needed)
- (h) preservation of FOL relation topology, which we call *relation footprint*, i.e. the ability to fully navigate the graph structure of n-ary relations instances, as described in Sect. 6.3.1
- (i) intuitiveness of representation and usability: these have been evaluated anecdotically and subjectively, but an ongoing study is addressing users. Indirect usability data come also from empirical studies on RDF temporal data patterns for linked data [42, 43], which we discuss in Section 6.5.1.

Dimensions (a) to (e) are quantitative and objective, dimensions (f) to (h) are objective, but qualitative, and dimension (i) is subjective and qualitative, except for combined results presented in Section 6.5.1. We represent qualitative dimensions on a three-value scale (good, limited, none).

In order to represent the approaches in a computable language, the patterns are exemplified with reference to a leading example of time indexing: the fact that Garibaldi landed in Sicily in 1860 (during the so-called *Expedition of the Thousand*). We restricted this in-vitro study to the simple case of ternary relations with a fixed temporal argument place, but we remark that larger arities are more complex variations of the same design patterns.

In order to create a critical mass of individuals and axioms for ontologies that sample the patterns, we have used SyGENiA (Synthetic GENerator of instance Axioms)³, which, contrarily to many random axiom generators, allows to specify a query that is used to generate ABox axioms from an ontology. We have used Pellet 2.3⁴ that produces accurate accounts of time used by the reasoner, running on a MacBookPro 3.06 GHz Intel Core Duo processor, with 8GB of RAM and MacOSX 10.7.3. We have not attempted to balance axioms with specific constructs, hoping that SyGENiA would think about it by using the queries provided. This is not completely true however, and the results, iterated 10 times to neutralize random effects of concurrent processes, reflect the lack of a real balance. Since we were interested in a first assessment of these dimensions, we accepted to live with some imbalance derived from the synthetic ABox, deferring a more grounded assessment to *in vivo* studies, e.g. [43].

In this section, the patterns are presented and exemplified quite schematically for space reasons, and their pros and cons with respect to the seven dimensions are briefly explained. Sect. 6.5 will recap on these summaries, and will present tables that compare the measures as well as the objective vs. subjective assessments.

6.4.1. Statement Contextualization

This pattern is one of the alternative OWL2 logical patterns for the FOL *sentence indexing* Hayes' logical pattern, à la context/modal logic or 3D ontologies. In RDF (*referring to a specific implementation by quad store using named graph*) or OWL2 a possible representation is depicted in Figure 6.2. This solution requires the contextualization of the triples using a certain index (1860 in the example). The contextualization can be implemented in different ways. In RDF, the triple data structure is extended by adding a fourth element to each triple, which is intended to express the *context* (a label), or the *named graph* (i.e. a document URI) including a set of triples. In OWL2, the same solution can be obtained by indicating an *ontology URI* instead of a named graph, either in a quad store for RDF serialization, or by using a *holdsAt* annotation property that asserts the time index to the named graph or the ontology URI. This solution has the advantage of being able to talk about assertions, but the disadvantages of needing a lot of contexts (i.e. documents) that could make a model very sparse.

³<http://code.google.com/p/sygenia/>

⁴<http://clarkparsia.com/pellet/>

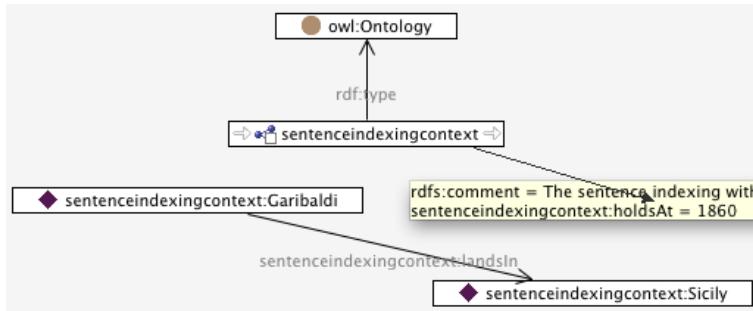


Figure 6.2. Example OWL graph for *StatementContextualization* pattern.

With respect to *polymorphism*, this solution cannot do much, because the possible design depends on the ability to create appropriate contexts that handle a partitioning of the triples that can prevent ambiguities, e.g. for different landing events of Garibaldi in Sicily in different years or with different modalities.

This solution disrupts the *relation footprint* depicted in Figure 6.1, because no connection is maintained between t and either the relation or the individuals. The sentence is actually transformed into a context (an ontology in this rendering), which is connected to t , but it is not available to DL reasoning. The footprint could be reconstructed in RDF by means of a SPARQL query, but only if we assume that each named graph only contains one occurrence of the `landsIn` property, and no other properties in the graph need to be indexed.

A recent additional solution to “make statements about statements” is the *singleton property* pattern (cf. [38] for an accurate comparison of expressivity and complexity of statement contextualization patterns), which reifies the instances of a property, uses them with punning as both properties and individuals, and relates them to the (intensional) property by means of the builtin property `rdf:sp`, e.g. `:landsIn#1 rdf:sp :landsIn . :Garibaldi :landsIn#1 :Sicily . :landsIn#1 :time “1860” ..` The singleton property pattern proves to be efficient, but it shares the same limitations of other statement contextualization solutions, as reported above.

A radical proposal to extend the RDF data structure to quintuples (“quins”) to deal with starting and ending times is [28], in which there is also a proof of the computational advantages of quin-stores over reified relation-based models.

6.4.2. StatementReification

This pattern is a second alternative for the *sentence indexing* Hayes’ logical pattern. In OWL2 (*OWL2 axiom reification/annotation*), or in RDF (*statement reification/annotation*) a possible solution is depicted in Figure 6.3. This solution requires reification of either a RDF statement or an OWL2 axiom, and its annotation. Syntactically, RDF and OWL2 differ, because the latter puts axiom reification and annotation in one construct. This solution has the advantage of

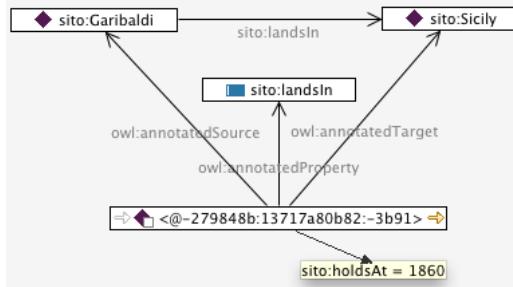


Figure 6.3. Example OWL graph for *StatementReification* pattern.

being able to talk about assertions, but the disadvantages of needing a lot of reification axioms, to introduce a primary binary relation to be used as a pivot for axiom annotations, and that in OWL2(DL) reasoning is not supported for axiom annotations.

With respect to *polymorphism*, this solution needs more statement/axiom annotations, e.g. for the sentence *Garibaldi landed in Sicily in 1860 with 1000 soldiers*:

_:[Garibaldi landsIn Sicily] with 1000Soldiers (6.7)

However, it is not possible to ensure that different events of Garibaldi landing in Sicily in the same year will be associated with the correct modality: if he landed twice, one with 500 soldiers, and another with 1000, both will be true for both landings.

This solution loosely maintains the *relation footprint*, because the connection between t and either the relation or the individuals is only granted by means of an annotation, or statement reification, which are not used for DL reasoning. Anyway, the footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

```
SELECT ?r ?x ?y ?t
WHERE {?r ?x ?y . ?a owl:annotatedProperty ?r .
      ?a owl:annotatedSource ?x .
      ?a owl:annotatedTarget ?y . ?a holdsAt ?t }
```

6.4.3. StatementAbstraction

This pattern is another alternative OWL2 logical pattern for the FOL *sentence indexing* logical pattern. A sample model on the running example is depicted in Figure 6.4. This solution requires a *singleton domain and range* for an object property, which is also punned as an individual in order to assert time indexing.

With respect to *polymorphism*, i.e. the need to add more arguments to the relation “on-the-go”, this solution needs more punned singleton-domain properties,



Figure 6.4. Example OWL graph for *StatementAbstraction* pattern.

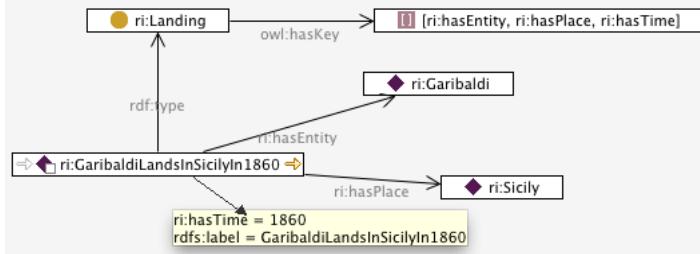


Figure 6.5. Example OWL graph for the *Situation* pattern.

e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

`GaribaldiLandsInSicilyWith1000Soldiers rdfs:subPropertyOf landsIn` (6.8)

`GaribaldiLandsInSicilyWith1000Soldiers rdfs:domain {Garibaldi}` (6.9)

`GaribaldiLandsInSicilyWith1000Soldiers rdfs:range {Sicily}` (6.10)

`GaribaldiLandsInSicilyWith1000Soldiers with 1000Soldiers` (6.11)

which looks quite goofy in terms of naming.

This solution loosely maintains the *relation footprint*, because the connection between t and either the relation or the individuals is only granted by means of the punning of the sub-property of `landsIn`, but punning does not enable DL reasoning across object and meta-level. Anyway, the footprint can still be reconstructed in RDF by means of a SPARQL query that traverses all the indirections created by this solution (domain and range to nominals to individuals, time to punned sub-property to property).

6.4.4. Situation

This pattern implements the *relation indexing* logical pattern, à la frame logic, situation calculus, or 3D+1 ontologies. A sample model on the running example is depicted in Figure 6.5. This solution requires to put `owl:hasKey` axioms for preserving identification constraints (i.e. that any two same occurrences of the relation would be actually inferred to be the same). Ordering can be put in the name of properties as an index when ambiguity arises. Punning is not needed. This solution has the advantage of being able to talk about assertions as (reifying) individuals, but the disadvantage of not being able to use them as properties as

well (this could be accommodated by punning the reified relation individuals as object or data properties).

Another advantage consists in representing temporally heterogeneous relations, at the cost of additional entities and axioms (one more property in the TBox, n more data declarations), e.g. for representing the sentence uttered in 2011: *John used to like 1899 New York*:

Liking hasAgent John (6.12)

Liking hasTime 2011 (6.13)

Liking hasTarget NewYork (6.14)

Liking hasTargetTime 1899 (6.15)

For a complete representation, there should be one main time (the time of the main fact expressed in the sentence), and other times related to situations referred by the elements of the sentence. E.g. in axiom 6.15 New York is considered at a different time from the time of the liking situation (as from axiom 6.14). The time of the sentence can even be different from the times reported in the content of that sentence, e.g. as from this multiple time reference uttered in 2013: *From 2007 to 2011, John used to like 1899 New York.*

With respect to *polymorphism*, this solution only needs more property assertions, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

GaribaldiLandsInSicilyIn1860 with 1000Soldiers (6.16)

This solution maintains the *relation footprint*, because the connection between t and the relationship is provided by a reification, which preserves direct connections to the entities, and ensures uniqueness of tuples via an `owl:hasKey` statement.⁵ This preserves DL reasoning and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL1.1 query with property paths,⁶ e.g.:

```
SELECT ?r ?x ?y ?t
WHERE {?r1 a ?r . ?r1 ?b1 ?x . ?r1 ?b2 ?y . ?r1 ?b3 ?t .
?r owl:hasKey/rdf:first ?b1 . ?r owl:hasKey/rdf:rest/rdf:first ?b2 .
?r owl:hasKey/rdf:rest/rdf:rest/rdf:first ?b3 }
```

The Situation pattern has been proposed in various forms. Noy et al. [37] describes a purely logical pattern; Gangemi et al. [14] proposes a **Situation** knowledge pattern⁷, with a basic OWL vocabulary, that has been extended in domain-oriented use cases.

⁵Note that OWL `hasKey` axioms (according to OWL 2 semantics) only apply to named individuals. Thus, identification constraints cannot be preserved if blank nodes are used as instances of the reified property.

⁶Property paths are not strictly needed, but the syntax is much more readable.

⁷Later implemented as <http://www.ontologydesignpatterns.org/cp/owl/situation.owl>

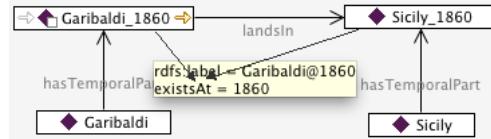


Figure 6.6. Example OWL graph for *Simple 4D* pattern.

6.4.5. Simple4D

This pattern implements a basic version of the *individual indexing* logical pattern, à la 4D ontology. As a sentence in FOL, an example is (cf. axiom schema 6.6, from which axiom 6.17 departs, because the binary function h cannot be expressed in OWL, and only the reified value of the function is therefore expressed). Figure 6.6 depicts a sample model on the running example for this pattern. It requires to introduce new entities: the time slices of the individuals referenced in the sentence.

$$\text{landsIn}(\text{Garibaldi}@1860, \text{Sicily}@1860) \quad (6.17)$$

An advantage of this solution is that it allows to compactly (no additional axioms required) represent temporally heterogeneous relations, e.g. *John@2011 likes NewYork@1899*. However, the time of the sentence as different from the times of the elements cannot be represented.

A general objection to 4D approaches is that we could have nD entities, e.g. for place, orientation, state of matter, numerosity, etc. In other words, we can perspectivize things within any dimension, not just time (cf. [12]). The solution 6.4.6 is an extension that allows to create more general *contextual slices*.

With respect to *polymorphism*, this solution needs more kinds of perspectivized individuals, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

$$\text{GaribaldiWith1000Soldiers with 1000Soldiers} \quad (6.18)$$

$$\text{Garibaldi hasCollectivePerspective GaribaldiWith1000Soldiers} \quad (6.19)$$

In this respect, 6.4.6 appears more elegant because it does not require additional names for each type of index, but just one for a context slice, while the context can bear the indices.

This solution maintains the *relation footprint*, because the connection between t and the relationship is a direct trickledown like in FOL. Also DL reasoning is preserved, and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

```

SELECT ?r ?x ?y ?t
WHERE { ?xt ?r ?yt . ?x hasTemporalPart ?xt .
?y hasTemporalPart ?yt . ?xt existsAt ?t . ?yt existsAt ?t }

```

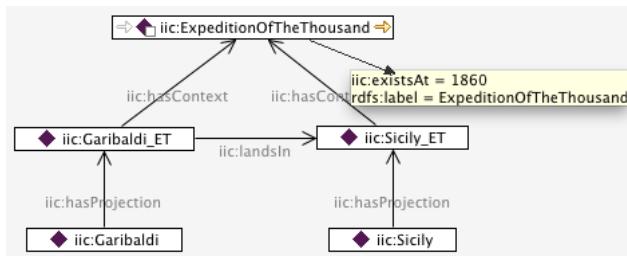


Figure 6.7. Example OWL graph for *Context slices* pattern.

6.4.6. ContextSlices

This pattern implements a more sophisticated variety of the *individual indexing* logical pattern, by actually mixing 6.4.4 and 6.4.5. As a sentence in FOL an example is:

$$\text{landsIn}(\text{Garibaldi}@1860, \text{Sicily}@1860, \text{ExpeditionOfTheThousand}) \quad (6.20)$$

that actually inserts new information by giving a name to a context parameter, i.e. the implicit knowledge of the context, in which Garibaldi landed in Sicily in 1860. In OWL2 (*contextualized individual logical pattern*), a possible solution is depicted in Figure 6.7. This solution requires to introduce new entities: the contextual projections and assignment of the individuals referred to in the sentence, as well as a context index that takes the indexes. An advantage of this solution is that it allows to get the freedom of the pattern 6.4.4, which allows to attach all kinds of indices to the reified relation (or context in this case), which overcomes e.g. the problem of dealing with multiple times, and the general perspectivization problem objected to 4D patterns.

With respect to *polymorphism*, this solution has the same simplicity as the 6.4.4 pattern: only one more axiom for each index is needed, attached to the context parameter. E.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

$$\text{ExpeditionOfTheThousand with 1000Soldiers} \quad (6.21)$$

However, when an index is bound to *parts* of a context, this solution needs to build a partonomy of contexts in order to deal with it. Such a partonomy may require a possibly elaborated reasoning in order to find out what is indexed in what part of a context, since some parts will be accessible by other parts, but others not. For example, *ExpeditionOfTheThousand* maximal context is a military campaign, in which Garibaldi's army numerosity has changed substantially through time, thus a *GaribaldiLanding* context that is part of *ExpeditionOfTheThousand* should be created, etc.

Similarly to pattern 6.4.5, this solution maintains the *relation footprint*, because the connection between t and the relationship is a direct trickledown. DL reasoning is preserved, and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

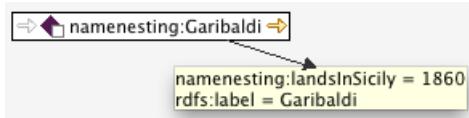


Figure 6.8. Example OWL graph for *Name nesting* pattern.

```
SELECT ?r ?x ?y ?t
WHERE { ?xt ?r ?yt . ?x hasProjection ?xt . ?c existsAt ?t .
?y hasProjection ?yt . ?xt hasContext ?c . ?yt hasContext ?c }
```

Context slices has been proposed e.g. by Chris Welty's⁸, based on the fluent representation described in [45]. A solution focused on temporal reasoning for OWL is presented in [5].

6.4.7. NameNesting

This solution implements the *name nesting* logical pattern, which results to be a common semantic web practice. As a sentence in FOL an example is:

$$\text{landsInSicily(Garibaldi,1860)} \quad (6.22)$$

while in OWL2 (*name nesting logical pattern*), a sample model on the running example is shown in Figure 6.8. This solution requires to nest the name of one of the arguments into the name of the property. Therefore, it works well when nested names are not essential to the knowledge to be represented. An advantage of this solution is clearly the minimal amount of axioms needed. However, the expressivity is very limited, and seems recommendable only in contexts when one argument is much more prominent than others that can be made implicit with name nesting, a form of “property localization”. A realistic example is the property `blogEntryDate`, which nests the *blog* argument in the property name: as far as we only talk of entry dates in a specific blog, it works. Disadvantages with respect to the other solutions include not being able to talk about the situation itself, and not being able to add further arguments.

With respect to *polymorphism*, this solution needs more localized properties, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

$$\text{Garibaldi landsInSicilyIn1860With 1000Soldiers} \quad (6.23)$$

Name nesting disrupts the *relation footprint*, because the connection between *t* and either the relationship or individuals is broken. DL reasoning is also very limited. The footprint cannot be reconstructed in RDF either.

6.5. Results and discussion

In the presentation of the patterns, we have summarized data about the four qualitative dimensions in the design space: DL reasoning support for full n-ary

⁸http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices

OWL pattern	DL reasoning	Polymorphism	Footprint	Intuitiveness
StatementContextualization	limited	none	none	good
StatementReification	limited	limited	limited	good
StatementAbstraction	limited	limited	limited	limited
Situation	good	good	good	good
Simple4D	good	limited	good	limited
ContextSlices	good	good	good	limited
NameNesting	limited/none	limited	none	good

Table 6.1. Qualitative evaluation of OWL patterns on a three-value scale (good, limited, none).

FOL pattern	OWL pattern	TBox axioms	ABox axioms	Annotation axioms	Total axioms
Sentence indexing (SI)	StatementContextualization	c	n	n	$2n + c$
Sentence indexing (SI)	StatementReification	c	$4n$	n	$5n + c$
Sentence indexing (SI)	StatementAbstraction	$4n$	n	0	$5n$
Relation indexing	Situation	c	$4n$	0	$4n + c$
Individual indexing	Simple4D	0	$7n$	0	$7n$
Individual indexing	ContextSlices	0	$9n$	0	$9n$
n/a	NameNesting	n	0	0	n

Table 6.2. Amount of OWL axioms needed for the representation of each pattern in function of the number n of sentences represented in the ontology.

sentences (f), polymorphism support (g), relation footprint (h), and intuitiveness (i): those data are also summarized in Table 6.1. The amount of axioms (a) dimension is reported in Table 6.2. The three purely computational dimensions: expressivity (b), consistency checking time (c), and classification time (d) are reported in Table 6.3. The last dimension: the amount of base and new constants (e) in the signature of each pattern, is reported in Table 6.4.

Among the nine design dimensions, the first five ((a) to (f)) (cf. data reported in Tables 6.2 and 6.3) are used to assess the *efficiency* and *expressivity* of a solution; (e) and (i) (cf. Tables 6.4 and 6.1) are used to assess its *usability*; (g) (cf. Table 6.1) is used to assess its *robustness and flexibility*; finally, (h) (cf. Table 6.1) is used to assess its potential for *interoperability*.

Concerning efficiency and expressivity, there is an uneven distribution of axioms, which e.g. in case of 1000 n-ary FOL sentences, would result in OWL models figures as different as 1000 for NameNesting to 9000 for ContextSlices pattern. StatementAbstraction is the only one to grow its TBox proportionally to the number of sentences, which we can expect to be computationally cumbersome, but actually that is not the case, at least on the base model.⁹ The theoretical expressivity remains low on all solutions, which is basically confirmed by empirical results of consistency checking and classification on the base model, where the only clear increase (about 300%) seems on the consistency checking time for the Situation pattern, which becomes worse with the artificial knowledge base, which has created 1000 artificial axioms and a variable number of individuals for each ontology implementing an example of a pattern. In terms of enabling DL reasoning over the entire n-ary sentence elements (cf. Table 6.1), three patterns are outstanding: Situation, ContextSlices, and Simple4D. This feature could actually correlate with longer times for reasoning tasks, although in this in vitro

⁹the artificial model cannot be generated for StatementAbstraction with SyGENiA because of the TBox generativity.

OWL Pattern	Expressivity	Cons Check Time Base	Classif. Time Base	Cons Check Time Artif	Classif. Time Artif	Artif Indiv	Artif Axioms
StatementContextualization	ELH	18	11	68	11	876	1012
StatementReification	ELH	18	14	64	10	617	1017
StatementAbstraction	ALHO(D)	18	7	n/a	n/a	823	1019
Situation	ALH(D)	54	21	1824	48	650	1024
Simple4D	ALH(D)	18	7	54	7	528	1027
ContextSlices	ALH(D)	19	26	57	23	551	1035
NameNesting	ALH(D)	17	7	75	6	1001	1007

Table 6.3. Expressivity of each OWL pattern and time taken for consistency checking and classification go the in-vitro knowledge base. Times are in *milliseconds*.

OWL Pattern	Class#base	Prop#base	Ind#base	NewIndividuals	NewProperties	NewClasses
StatementContextualization	0	2	2	0	0	0
StatementReification	1	5	2	0	0	0
StatementAbstraction	1	3	3	n	n	2n
Situation	2	3	3	n	2	1
Simple4D	1	2	4	2n	1	0
ContextSlices	2	4	5	3n	2	2
NameNesting	1	1	1	-1n	0	0
FOL	0	1	3	0	0	0

Table 6.4. Data about the constants needed for each OWL pattern. Reference FOL data is in the last row.

study such correlation is not completely proved.

Concerning usability, on one hand intuitiveness is highly debated. A common assumption is that individual-indexing-based patterns are less intuitive, because humans tend to reason by attributing some “endurant” qualities to objects, rather than thinking about them in terms of “slices”. As [20] correctly notices, this could be due to cultural effects, since many scientists and manufacturers reason exactly in terms of slices, let alone the possibility that some features of noun-centric grammars of Western languages may induce an endurant effect. However, the ContextSlices pattern seems to be a good compromise, by minimizing the slices to the situations or contexts where objects are considered. On the other hand, the need to generate new terms in the signature of either a TBox or ABox is usually considered problematic for intuitiveness. Humans are accustomed to consider a conventionally finite amount of entities, and making them grow quite artificially with reification, abstraction, or slicing reduces usability. Also, semantic web toolkits are not quite proficient in smoothing the process of creating additional entities. Finally, linked data and semantic web developers are quite suspicious of entities that are created for the sake of design, rather than actually referenced in databases, texts, or social networks. The best patterns from this respect are StatementContextualization, StatementReification, and NameNesting, which seem actually the first choice in lightweight vocabularies and linked data, and typically preferred in the loose debates from the semantic technology blogosphere.

Concerning robustness/flexibility, polymorphism is poorly supported by 5 patterns out of 7. Only Situation and ContextSlices are flexible enough to make their knowledge bases grow linearly when new arguments are added to an n-ary relation. Situation is probably ideal here, because it is isomorphic to linguistic

frame semantics, which is assumed to be closest to the cognitive structures that humans use in organizing their knowledge (cf. e.g. [12]).

Concerning interoperability, the relation footprint from the FOL sentences is still there in 5 patterns, which is good news for future unification algorithms: we have shown some simple SPARQL queries to extract the footprint from some patterns, and we can generate them from the footprint as well, so enabling a lightweight unification. Unfortunately, relation footprint tends to disappear when expressivity decreases, as is the case with StatementContextualization and NameNesting.

6.5.1. Comparison to LOD data patterns evidence

Additional dimensions to the comparison between n-ary modeling patterns need to be added from empirical analysis of actual usage and subjective preference or intuitivity of the patterns. Leaving the second to ongoing experiments, for the first, we have some recent results that can be combined to our results. The survey presented in [42] contains data about the usage of temporal information in Linked Open Data. They present 5 design patterns, represented over an abstraction of the RDF data model. There is an interesting and useful overlap with the patterns presented here: the authors distinguish between document-centric and fact-centric temporal perspectives on representing temporal information in RDF, the second being the one addressed in this paper as well. Among fact-centric patterns, they identify two “sentence-centric” patterns: *Reification* and *Applied temporal RDF*, and two “relationship-centric” patterns: *N-ary relationship*, and *4D-fluents*. Their *Reification* corresponds to our *StatementReification* pattern, their *Applied temporal RDF* corresponds to our *StatementContextualization* pattern, their *N-ary relationship* corresponds to our *Situation* pattern, and their *4D-fluents* corresponds to our *Simple4D*, although the authors attribute their *4D-fluents* explicitly to [45], which is actually our *ContextSlices* pattern, but their example does not provide a context index, then making it a case of *Simple4D* instead. They do not search data patterns for *StatementAbstraction*, *ContextSlices*, and *NameNesting*. The quantitative results of [42] indicate a neat preference of LOD for two patterns: the document-centric pattern called *Metadata-based representation*, which looks quite close to our *StatementContextualization* pattern (as well as to their *Applied temporal RDF* pattern), and the *N-ary relationship* (our *Situation*) pattern. The authors of [42] do not give data for *NameNesting*, which seems anecdotically well attested in LOD.

6.5.2. General assessment

As a general assessment, more sophisticated patterns stand out on most criteria, however potential (anecdotically based) usability aspects and avoidance of newly introduced entities may work for the “unsophisticated” solutions such as StatementContextualization and NameNesting, which are however also the least interoperable patterns. Usage data from [42] are also ambivalent, since by far the most used data patterns are *Situation* and *StatementContextualization*. It is therefore easy to recognize, on objective grounds, the gap (cf. [24]) that has

emerged when the linked data wave has hit the neat island of logically sound and well-designed ontologies. We can only hope for a reduction of the gap, partly with automatic reconciliation techniques, partly with the good practice of listening to the reasons and practices of the different parties.

6.6. Intensional multigrade predicates and D&S

When moving to intensional multigrade predicates, we shift the scope of ontology design patterns from a universe of “typical” entities to a universe of concepts. As mentioned in the introduction, we often have to deal at the same time with both representation of knowledge about the world, and knowledge about the way we construct the world. A good example is when a physical event, e.g. a boy quickly grabbing a bag, is interpreted in different ways by different observers: a) the boy is a traveller almost missing a flight, b) the boy is a robber, c) the boy is cute (by a girl appreciating his smooth movements), d) the bag looks heavy but the boy strong, etc. For each interpretation, we have the same elements: a boy, a bag, some movements, the qualities of those objects and events, etc. But those elements are selected and arranged differently through the lenses of the *framing* (or *description*) applied by the observer.

For space reasons, we only present here the ontology design patterns related to D&S, which cover the spectrum of frames and their application to situations. Notice that the patterns presented in the previous sections, in particular ContextSlices, can be used jointly with D&S.

Descriptions and Situations (D&S) [11, 15] is an ODP framework aimed at integrating the representation of the extensional and intensional layers of multi-grade predicates. It is implemented in different ways, but the minimal one¹⁰ is a knowledge pattern usable as a representation framework for several complex patterns. The basic intuition goes to the ability to distinguish two levels of description:

- a “ground” description, using whatever vocabulary or axioms, which constitutes the representation of a **situation**. The situation section of D&S is basically a container for vocabulary and axioms that help refactoring a FOL n-ary (typically ≥ 2) relation into a class with n binary relations, i.e. the so-called *n-ary relation (logical) pattern*
- a “duper” description, using the D&S vocabulary or some specialization of it, which provides an alternative schema to describe a class of situations (that already have a ground description, or that can get one emerging out of existing data). The description part of D&S talks about (reified) relations, concepts, and the relations between them and between them and ground entities.

6.6.1. Sample topics and sentences

These topics and examples are taken from the DeepKR challenge program¹¹:

¹⁰<http://www.ontologydesignpatterns.org/descriptionandsituation.owl>

¹¹<https://sites.google.com/site/dkrckcap2011/>

1. High level descriptions: e.g. “Enzymes proofread DNA during its replication and repair damage in existing DNA”
2. Explanation: “The logistics of carrying out metabolism set limits on cell size.”
3. Hypotheticals, particularly as parts of explanations: e.g. “The lysosome provides a space where the cell can digest macromolecules safely, without the general destruction that would occur if hydrolytic enzymes roamed at large.”
4. Representation of context: e.g. “In adults, the illness is rarely serious, but in infants it can be fatal”. “In the elderly, pneumonia is a common cause of confusion.”

Similarly, but from D&S story since 2003:

1. Interpretation of situations (e.g. in archaeology, literature, psychology)
2. Diagnostic interpretation of cases (diagnosis of diseases and syndromes against signs and findings)
3. Normative interpretation of cases (case abstraction, application of norms and meta-norms in Law)
4. Plan models to be satisfied on scheduling or execution
5. Plan models or schedules to be built based on available resources
6. Control checking (matching actual vs. expected facts, classifying unwanted or unexpected facts)

6.6.2. Some approaches to talk about descriptions

Descriptions are typically made in a logical language. For example, all solutions exemplified in the previous sections are descriptions of some facts or situations. Following logical good practices for preserving the integrity and complexity results about first-order languages, the vocabulary used to design those descriptions is disjoint from the vocabulary of the individuals described. Many workarounds have been proposed for mixing different logical layers without needing actual Higher-Order Logic (HOL), e.g. OWL2 punning [36], Hi-DL-Lite [9], Hayes’ *pollarding* in Common Logic, and Kifer’s F-Logic [26].

However, logical approaches do not provide any good practices to use such expressive capabilities, i.e. what such additional expressivity should be used for. Some use cases and solutions have been provided e.g. by the W3C task force on ontology patterns [37], limited to the capability of talking about concepts, for example when representing subjects of books. D&S [15] was the first proposal of a formal ontology for talking jointly about concepts and (reified) relations. A detailed axiomatization, based on DOLCE, of a part of D&S is [30], cf. also [44] in this book. A lightweight example of an ontology for talking about concepts and the relations between them is SKOS [32]. LMM [12] is a semiotic ontology to

talk about the meaning and reference of linguistic expressions, and makes wide usage of punning axioms (cf. the chapter on information object patterns in this book).

Departing from the DOLCE-oriented version of D&S, cDnS [11] is an attempt to cover most of the entities that lie in the grey area between typical, publicly recognized individuals (e.g. physical objects), and typical, publicly recognized predicates (e.g. classes, relations): concepts, collections, frames, communities, terms, meanings, interpretations (plans, norms, diagnoses, hypotheses, etc.).

Here we drastically simplify the problems in the literature, and present some simple patterns to:

- create a vocabulary for the intension of concepts and (reified) relations
- talk about descriptions, i.e. to *redescribe*, or *dupe* (in the sense of duplicating) a logical description (e.g. an assertional axiom), hence the name “duper”
- represent such vocabulary by exploiting OWL2 punning, i.e. *super-dupering* logical descriptions
- adding RIF rules in order to generalize the pattern

6.6.3. The Description pattern

The **Description** pattern¹² is very simple, only providing the means to associate descriptions (reifications of the intension of n-ary relations) with the concepts they either define or use, and a specialization property between either concepts or descriptions:

OWL Classes: *Description* (*D*), *Concept* (*C*)

OWL ObjectProperties: *defines* ($\subseteq D \times C$), *usesConcept* ($\subseteq D \times C$), *specializes* ($\subseteq (D \times D \cup C \times C)$)

Examples:

ItalianConstitutionalLaw rdf:type D (6.24)

Citizen rdf:type C (6.25)

ItalianConstitutionalLaw
defines Citizen (6.26)

ItalianConstitutionalLaw
usesConcept Republic (6.27)

ParliamentaryRepublic
specializes Republic (6.28)

The advantages of the **Description** pattern are in the possibility of (lightly) talking about complex entities such as plans, norms, diagnoses, hypotheses, desires, etc., which are relational in nature, without resorting to modal logics. When a deep reasoning on these entities is required, we can wrap some specialized reasoner, which will benefit from the explicit typing of the entities to be treated in its logical language (e.g. a deontic logic or a constraint satisfaction language).

¹²<http://www.ontologydesignpatterns.org/cp/owl/description.owl>

Description also nicely complements SKOS, since *Concept* can be aligned to *skos:Concept*, and *specializes* to *skos:broader*, while SKOS lacks notions like *Description* and *defines*.

Consider that **Description** alone is not addressing any of the Deep KR tasks mentioned in the Problem section. Next section is about full D&S, which addresses those tasks.

6.6.4. The D&S basic pattern

Traditionally, the universe of facts is not the same as the universe of concepts. With multiple descriptions, clarifying the disjoint universes is specially important in a reified world. In the past, KR deep reasoning tasks were dealt with specialized languages, but in an integration perspective we need clear universes of discourse. Some of those tasks address:

- legal knowledge (facts vs. cases vs. norms vs. meta-norms)
- services, planning and control (actual vs. expected facts)
- diagnoses and situation awareness (ground vs. (un)wanted facts)
- hypothesis testing and CBR (building/testing interpretations from facts)

The **Descriptions and Situations** pattern¹³ (D&S) is a pattern to represent the entities involved in a common abstract task: *Classifying a situation*, according to possibly incomplete, alternative or loose constraints and concepts. When classifying a situation, constraints must be explicit and explicitly linked to the representation of a situation. D&S enables the representation of situations and descriptions, and their respective links: entities of a situation that play roles from a description, values of a situation that fit parameters from a description, events of a situation that accomplish a task defined by a description, etc. Given its general intuition, representing and reasoning with D&S gets to grips with its possible implementations, which are summarized in the following sections.

6.6.5. The *segregated* D&S “duper” pattern

The first implemented versions of D&S were written in DAML+OIL and OWL1, where no punning is possible. Duping the concepts in the signature of the ontology was therefore needed. The additional axioms needed to abridge the **Situation** (S) and *Description* (D) axioms are the following:

OWL ObjectProperties: *classifies* ($\subseteq (Concept \times owl:Thing \cup Description \times Situation)$)

An **example** ontology for this pattern is the following: [Generic TBox addons]

$$\text{CoreConcept} \text{ rdfs:subClassOf } \text{C} \quad (6.29)$$

¹³<http://www.ontologydesignpatterns.org/cp/owl/descriptionandsituation.owl>

[Domain TBox]

SalesModel rdf:type D (6.30)

RegularSale rdfs:subClassOf S (6.31)

RegularSale rdfs:subClassOf
(classifies⁻ value SalesModel) (6.32)

RegularSale owl:equivalentClass
(isSettingFor some (classifies⁻
(CoreConcept and some
(defines⁻ SalesModel))) (6.33)

[ABox]

SalesModel defines
{Seller, Buyer, Product, SaleTime} (6.34)

Seller rdf:type CoreConcept (6.35)

Buyer rdf:type CoreConcept (6.36)

Product rdf:type CoreConcept (6.37)

SaleTime rdf:type C (6.38)

Sale_{c4598} isSettingFor
(Apple, Mustafa, iPad) (6.39)

Seller classifies Apple (6.40)

Buyer classifies Mustafa (6.41)

Product classifies iPad (6.42)

[Inferred axioms]

$\models \text{Sale}_{c4598} \text{ rdf:type RegularSale}$ (6.43)

$\models \text{Sale}_{c4598} \text{ classifies}^{-} \text{SalesModel}$ (6.44)

Duper D&S is good at representing the links between entities from a situation and concepts from a description, but as far as reasoning is concerned, the main tasks that can be carried out are limited to the knowledge bases where both situations and descriptions are sufficiently described and linked: representations of contexts (such as in the DeepKR scenario of age-oriented contextualization of diseases), executed plans with reference to plan models, applied (e.g. verified) interpretations and hypotheses, etc. Even in these cases, only some simple tasks can be performed in OWL(1).

In the example above, we are able to infer that a certain sale is an instance of *RegularSale*, and even that that sale is classified by the description *SalesModel*, but this inference power is quite weak, because we cannot state that *all and only* the things in the setting that are classified by a core concept that is defined by *SalesModel* can make the situation classified under *SalesModel*: we can say “some” of them; in practice, a situation with just one seller is enough to classify it as *RegularSale*!

In order to circumvent this problem, we might include explicitly all classification restrictions for core concepts in the equivalence axiom (6.45).

```
RegularSale owl: equivalentClass
  (isSettingFor some
    (classifies- some (Seller and
      (defines- value SalesModel))))
etc. (6.45)
```

This solution is really ad hoc, and definitely ugly. It'd be tempting here to use directly the **Situation** pattern alone, so as to create a catch-it-all *owl:equivalentClass* axiom (6.46).

```
RegularSale owl: equivalentClass
  (isSettingFor some Seller))
  (isSettingFor some Buyer))
  (isSettingFor some Product)) (6.46)
```

However, the issue here is being able to *take into account also axioms made at the description layer*, such as the fact that some concepts are core, and others not, or that Seller is dual to Buyer (i.e. they are mutually dependent for a regular sale transaction). Moreover, how to state e.g. that *Seller* is a *CoreConcept* and a dual to *Buyer* only *in the context of a RegularSale*? In practice, we need equivalentClass axioms that are generalized to one holding for all classes of situations. E.g. something like the FOL axiom 6.47:

$$\begin{aligned} \forall(s, e, c, d)((\\ isSettingFor(s, e) \wedge CoreConcept(c) \wedge \\ classifies(c, e) \wedge defines(d, c) \wedge \\ \neg\exists(e_1, c_1)(\\ isSettingFor(s, e_1) \wedge \neg CoreConcept(c_1) \wedge \\ \neg classifies(c_1, e_1) \wedge \neg defines(d, c_1))) \\ \rightarrow (classifies(d, s) \wedge S(s))) \end{aligned} \quad (6.47)$$

Additionally, with Duper D&S we are obliged to (manually or programmatically) create two distinct constants for the description (*SalesModel*), and the situation class (*RegularSale*), which are the intensional respectively extensional aspects of the same (reified) relation.

Indeed, with Duper D&S it's very hard to figure out not only the following 2, 3, and 4 tasks, which feature incomplete information, so that soft, fuzzy or probabilistic rules would be possibly needed, but even 1, which is a deterministic task:

1. to decide if a situation can be (partly or fully) classified under a description by classifying relevant things in relevant concepts, in presence of description- or concept-level relations (diagnoses, control checking)

2. to evaluate what situation fits best a description (discovery, knowledge mining)
3. to make a situation emerge out of scattered facts (case-based reasoning, heuristic classification)
4. to evaluate what description fits best in order to unify a certain set of facts (norm application, hypothesis testing, explanation)

In the next section, we show novel patterns that employ OWL2 and RIF in order to solve tasks of type (1). Tasks of the other types would require soft rules (4), or hybridizing deterministic and probabilistic reasoners (2,3), whose patterns are not covered here.

6.6.6. Super-duper D&S

In OWL2, the *equivalentClass* axiom 6.33 could be more elegantly implemented with a property chain (axiom 6.48), while the duplicated vocabulary issue can be solved via punning, so that the overall model becomes as follows:

[D&S TBox add-ons]

$$\text{(isSettingFor} \circ \text{classifies}^- \circ \text{defines}^-) \text{owl: subPropertyOf classifies}^- \quad (6.48)$$

$$\text{CoreConcept rdfs: subClassOf C} \quad (6.49)$$

[Domain TBox]

$$\text{RegularSale rdf: type D} \quad (6.50)$$

$$\text{RegularSale rdfs: subClassOf S} \quad (6.51)$$

$$\begin{aligned} &\text{RegularSale rdfs: subClassOf} \\ &\text{(classifies}^- \text{ value RegularSale)} \end{aligned} \quad (6.52)$$

[ABox]

$$\text{Seller rdf: type CoreConcept} \quad (6.53)$$

$$\text{Buyer rdf: type CoreConcept} \quad (6.54)$$

$$\text{Product rdf: type CoreConcept} \quad (6.55)$$

$$\text{SaleTime rdf: type C} \quad (6.56)$$

$$\text{(Seller dualTo Buyer))} \quad (6.57)$$

$$\text{RegularSale defines}$$

$$\{\text{Seller, Buyer, Product}\} \quad (6.58)$$

$$\text{Sale}_{c4598} \text{ isSettingFor} \quad (6.59)$$

$$(\text{Apple, Mustafa, iPad})$$

$$\text{Seller classifies Apple} \quad (6.60)$$

$$\text{Buyer classifies Mustafa} \quad (6.61)$$

$$\text{Product classifies iPad} \quad (6.62)$$

[Inferred axioms]

$$\models \text{Sale}_{c4598} \text{ rdf:type RegularSale} \quad (6.63)$$

$$\models \text{Sale}_{c4598} \text{ classifies}^- \text{ RegularSale} \quad (6.64)$$

Notice the “mapping” axiom (6.52) of *RegularSale* on being classified by itself. A variant was needed in OWL1 Duper D&S (axiom 6.33) in order to connect *RegularSale* to *SalesModel*, and making the classification be inherited to instances of *RegularSale*. However, even in OWL2 SuperDuper D&S it is needed, because punning does not substantially change the semantics of the OWL1 ontology.

For completeness, we might mention that a purely “situational” solution can be provided in OWL2, by adding punning axioms such as 6.65, 6.66, and 6.67 directly to axiom 6.46:

$$(\text{Seller} \text{ rdf:type CoreConcept}) \quad (6.65)$$

$$(\text{Seller} \text{ dualTo Buyer}) \quad (6.66)$$

$$(\text{Seller} \text{ defines}^- \text{ RegularSale}) \quad (6.67)$$

In practice, we would reconstruct the Description-level axioms as in the SuperDuper sample ABox above (axioms 6.53, 6.57, 6.58, but they would be semantically disjoint from the Situation-level axioms.

Therefore, besides being cleaner, the SuperDuper OWL2 implementation of the pattern does not provide a generalized solution as required by task (1). Nonetheless, punning allows us to easily hybridize OWL with RIF, as shown in the next section.

6.6.7. Super-duper D&S + RIF

The last solution proposed here hybridizes the OWL2 version of the D&S pattern with a RIF rule. Rule 6.68 makes, on a global base, a rule reasoner infer the classification axiom between a situation and a description whereas all core concepts defined by the description are classified by entities in the setting of a situation. The consequent part of the rule also makes us infer that that description is subclass of the class *Situation*. The double interpretation of the description can

be made thanks to punning.

```

forall ?s ?e ?c ?d
? s [rdf:type ?d] |
? d [rdfs:subClassOf Situation]
←
and (
? s [isSettingFor ?e]
? e [rdf:type ?c]
? c [defines- ?d]
? c [rdf:type CoreConcept]
(not (exists ?e1
and (
? s [isSettingFor ?e]
(not (? e [rdf:type ?c])))
(not (? c [defines- ?d])))))
(6.68)

```

The RIF rule presented here lets us accomplish some tasks of type (1), since it complements the OWL2 pattern to reach the expressivity of the axiom 6.47. The most important novelty with respect to previous D&S implementations is that the SuperDuper schema puns the names of descriptions and situation types (the intensional vs. extensional parts of a reified n-ary relation), and cleanly links to rules 6.68 in order to making *rdf:type* and *classifies-* isomorphic in the scope of the D&S pattern.

6.7. Conclusion

In this review of design patterns for representing situations, and the interplay between descriptions and situations, we have shown that a classic reified approach to represent situations, together with description-level axioms, can be used as a knowledge pattern library (a.k.a. *pattern language*) to represent extensional and intensional multigrade predicates. Such a pattern requires a rich expressivity in order to accomplish even basic useful reasoning tasks. The solution proposed includes some novel constructs introduced by OWL2 (punning, keys, property chains), as well as a rule language like RIF Core, which extends SWRL towards FOL with fully quantified rules.¹⁴ Other rule languages could be used for this task, so that the solution introduced can be also used as a bridge between DL-based ontologies, and AI languages such as KM, OCML, PowerLoom, Alchemy, etc. In particular, we have tested OCML, and it allows a clean implementation of the RIF rule.

¹⁴Such an extension of SWRL was proposed already in 2005 within W3C, cf. <http://www.w3.org/Submission/SWRL-FOL>.

6.8. The ODP portal

The ODP wiki¹⁵ contains support for collecting, reviewing, annotating and publishing knowledge patterns like the ones introduced here [6, 16, 41].

Bibliography

- [1] C. Alexander. *The Timeless Way of Building*. Oxford Press, 1979.
- [2] F. Baader, editor. *The Description Logic Handbook: theory, implementation, and applications*. Cambridge University Press, Cambridge, 2003.
- [3] E. Bach. On Time, Tense, and Aspect: An Essay in English Metaphysics. In *Radical Pragmatics*, 1981.
- [4] C. Baker, C. Fillmore, and J. Lowe. The Berkeley FrameNet Project. In *Proceedings of the 17th international conference on Computational linguistics*, pages 86–90, Morristown, NJ, USA, 1998.
- [5] S. Batsakis and E. Petrakis. SOWL: A Framework for Handling Spatio-Temporal Information in OWL 2.0. In *5th International Symposium on Rules: Research Based and Industry Focused (RuleML 2011)*, 2011.
- [6] E. Blomqvist, V. Presutti, E. Daga, and A. Gangemi. Experimenting with extreme design. In *EKAW*, pages 120–134, 2010.
- [7] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. In *WWW 2005, May 10-14, 2005, Chiba, Japan*. ACM, 2005.
- [8] D. Davidson. The Logical Form of Action Sentences. In *The Logic of Decision and Action*. University of Pittsburgh Press, Pittsburgh, 2nd edition, 1967.
- [9] G. De Giacomo, M. Lenzerini, and R. Rosati. Towards Higher-Order DL-Lite. In *Proc. of DL2008*, Dresden, Germany, 2008.
- [10] A. Galton. Time and Change for AI. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 4, pages 175–240. Clarendon Press, Oxford, 1995.
- [11] A. Gangemi. Norms and Plans as Unification Criteria for Social Collectives. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(3), 2008.
- [12] A. Gangemi. What's in a schema? a formal metamodel for egc and framenet. In C.-R. Huang, N. Calzolari, A. Gangemi, A. Lenci, A. Oltramari, and L. Prévôt, editors, *Ontology and the Lexicon, Studies in Natural Language Processing*. Cambridge University Press, 2010.
- [13] A. Gangemi. Super-duper schema: an owl2+rif dns pattern. In V. Chaudry, editor, *Proceedings of DeepKR Challenge Workshop at KCAP11*, 2011.
- [14] A. Gangemi and P. Mika. Understanding the Semantic Web through Descriptions and Situations. In *CoopIS/DOA/ODBASE*, pages 689–706, 2003.
- [15] A. Gangemi and P. Mika. Understanding the Semantic Web through Descriptions and Situations. In *CoopIS/DOA/ODBASE*, pages 689–706, 2003.
- [16] A. Gangemi and V. Presutti. Ontology Design Patterns. In S. Staab and R. Studer, editors, *Handbook on Ontologies, 2nd Edition*. Springer Verlag, 2008.

¹⁵<http://www.ontologydesignpatterns.org>

- [17] A. Gangemi and V. Presutti. Towards a pattern science for the semantic web. *Semantic Web*, 1(1-2):61–68, 2010.
- [18] A. Gangemi and V. Presutti. A multi-dimensional comparison of ontology design patterns for representing n-ary relations. In *SOFSEM 2013: Theory and Practice of Computer Science*, pages 86–105. Springer, 2013.
- [19] G. Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, Enschede, The Netherlands, Enschede, October 2005.
- [20] P. Hayes. Formal unifying standards for the representation of spatiotemporal knowledge. Technical report, IHMC, 2004. <http://www.ihmc.us/users/phayes/Trickledown2004.pdf>.
- [21] D. Hernández, A. Hogan, and M. Krötzsch. Reifying RDF: what works well with wikidata? In T. Liebig and A. Fokoue, editors, *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*, volume 1457 of *CEUR Workshop Proceedings*, pages 32–47. CEUR-WS.org, 2015.
- [22] J. R. Hobbs. Ontological Promiscuity. In *Proceedings, 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.
- [23] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, to appear.
- [24] P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, and A. P. Sheth. Linked Data is Merely more Data. In *Proceedings of the AAAI Symposium Linked Data Meets Artificial Intelligence*, pages 82–86, 2010.
- [25] H. Kamp and U. Reyle. From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. In S. Link and H. Prade, editors, *Foundations of Information and Knowledge Systems*, volume 42 of *Studies in Linguistics and Philosophy*. Kluwer / Dordrecht, 1993.
- [26] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of Association for Computing Machinery*, 42:4:741–843, 1995.
- [27] W. Köhler. *Gestalt Psychology*. Liveright, New York, 1947.
- [28] H.-U. Krieger. A Temporal Extension of the Hayes/ter Horst Entailment Rules and an Alternative to W3C’s N-ary Relations. In *International Conference on Formal Ontology and Information Systems (FOIS 2012)*, 2012.
- [29] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino. Social roles and their descriptions. In C. Welty and D. Dubois, editors, *Proc. of Knowledge Representation and Reasoning (KR)*, pages 267–277, 2004.
- [30] C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino. Social roles and their descriptions. In C. Welty and D. Dubois, editors, *Proc. of Knowledge Representation and Reasoning (KR)*, pages 267–277, 2004.
- [31] J. McCarthy. Actions and other events in situation calculus. In *Proceedings of KR-2002*, 2002.
- [32] A. Miles and D. Brickley. SKOS Core Vocabulary Specification. Tech-

- nical report, World Wide Web Consortium (W3C), November 2005.
<http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/>.
- [33] M. Minsky. A Framework for Representing Knowledge. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.
 - [34] M. S. Moore. Legal Reality: A Naturalist Approach to Legal Ontology. *Law and Philosophy*, 21, 2002.
 - [35] B. Motik. On the properties of metamodeling in OWL. *J. Log. Comput.*, page 617–637, 2007.
 - [36] B. Motik. On the properties of metamodeling in owl. *J. Log. Comput.*, 17(4):617–637, 2007.
 - [37] Natasha Noy and Alan Rector. Defining N-ary Relations on the Semantic Web: Use With Individuals. Technical report, W3C, 2005.
<http://www.w3.org/TR/swbp-n-aryRelations/> (2004).
 - [38] V. Nguyen, O. Bodenreider, and A. Sheth. Don't like rdf reification?: making statements about statements using singleton property. In *Proceedings of the 23rd international conference on World wide web*, pages 759–770. ACM, 2014.
 - [39] A. G. Nuzzolese, A. Gangemi, and V. Presutti. Gathering Lexical Linked Data and Knowledge Patterns from FrameNet. In *Proc. of the 6th International Conference on Knowledge Capture (K-CAP)*, pages 41–48, Banff, Alberta, Canada, 2011.
 - [40] A. Oliver and T. Smiley? Multigrade Predicates? *Mind*, 113, 2004.
 - [41] V. Presutti, E. Daga, A. Gangemi, and A. Salvati. [http://ontologydesign-patterns.org \[odp\]](http://ontologydesign-patterns.org [odp]). In *International Semantic Web Conference (Posters & Demos)*, 2008.
 - [42] A. Rula, M. Palmonari, A. Harth, S. Stadtmuller, and A. Maurino. On the Diversity and Availability of Temporal Information in Linked Open Data. In *Proceedings of ISWC2012*, 2012.
 - [43] A. Scheuermann, E. Motta, P. Mulholland, A. Gangemi, and V. Presutti. An empirical perspective on representing time. In *Proceedings of the seventh international conference on Knowledge capture*, pages 89–96. ACM, 2013.
 - [44] Valentina and A. Gangemi. Dolce+d&s ultralite and its main ontology design patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. In this volume, chapter 5. IOS Press/AKA Verlag, 2016.
 - [45] C. Welty and R. Fikes. A Reusable Ontology of Fluents in OWL. In *Int. Conference on Formal Ontology and Information Systems*. IOS Press, 2006.

This page intentionally left blank

Chapter 7

Ontology Pattern Languages

Ricardo Falbo, Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil

Monalessa Barcellos, Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil

Fabiano Ruy, Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória; Informatics Department, Federal Institute of Espírito Santo, Campus Serra, Serra, Brazil

Giancarlo Guizzardi, Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil

Renata Guizzardi, Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil

Ontology design patterns are a promising approach for ontology engineering. In this chapter, we introduce the notion of Ontology Pattern Language (OPL) as a way to organize domain-related ontology patterns. This chapter is organized as follows: Section 7.1 presents the motivation for organizing Domain-Related Ontology Patterns (DROPs) as OPLs. Section 7.2 discusses what an OPL is, and how OPLs are represented, showing an example of an OPL for the software process domain, based on ISO Standards (ISP-OPL). Section 7.3 discusses how to build OPLs from core ontologies, taking ISP-OPL as an example. Section 7.4 discusses how an OPL can be used for building a domain ontology. An example applying ISP-OPL for building a domain ontology about the Stakeholder Requirements Definition Process is presented. Section 7.5 presents an overview of existing OPLs. Finally, Section 7.6 presents our final remarks.

7.1. Motivation

An Ontology Pattern (OP) describes a particular recurring modeling problem that arises in specific ontology development contexts and presents a well-proven solution for the problem. Using OPs is an emerging approach that favors the reuse of encoded experiences and good practices. Different kinds of OPs support ontology engineers on distinct phases of the ontology development process [7]. For instance, a *Domain-related Ontology Pattern* (DROP) is a kind of *Content Ontology Design Pattern* that captures a reusable fragment extracted from a reference domain ontology, which may assist in building the conceptual model of a new domain ontology. A *Logical Ontology Design Pattern*, or an *Ontology Coding Pattern*¹, in turn, supports ontology implementation, aiming at solving problems related to reasoning or related to shortcomings in the expressivity of a specific logical formalism.

This chapter focuses on the conceptual modeling phase of ontology engineering, thus dealing with DROPs. To emphasize this point, it is important that the reader understands that in this chapter, in an analogy to Software Engineering, we take an *analysis* point of view, in which we build a model of the world, without any implementation concerns (e.g. tractability or efficiency). Typically, this conceptual model will be later refined and, finally, become an *ontology schema* after some adaptations required by a specific ontology modeling language/formalism.

As pointed out by Alexander and colleagues in their pioneering work [1], each pattern can exist only to the extent that it is supported by other patterns. According to Schmidt et al. [25], in the community of Software Engineering (SE) patterns, the trend is towards defining pattern languages, rather than stand-alone patterns. The term “pattern language” in SE refers to *a network of interrelated patterns that defines a process for systematically solving coarse-grained software development problems* [5] [3]. This approach can also be taken into account in Ontology Engineering, giving rise to **Ontology Pattern Languages** (OPLs).

Although many DROPs in the literature refer to other patterns, and explore relations such as subsumption and composition, most of these references fail to give more complete guidelines on how the patterns can be combined to form solutions to larger problems. Context and problem descriptions are usually stated as general as possible, so that each pattern can be applied in a wide variety of situations. In addition, solution descriptions tend to focus on applying the patterns in isolation, and do not properly address issues that arise when multiple patterns are applied in overlapping ways, such as the order in which they can be applied. This approach is insufficient for complex real-world situations, since the features introduced by applying one DROP may be required by the next. A larger context is therefore needed to describe larger problems that can be solved by combining patterns, and to address issues that arise when DROPs are used in combination [6].

It is important to highlight that Ontology Engineering is a complex task. It is of course important to take into consideration the need for speedy and easy development, and this is indeed another reason for motivating **reuse** in this area. However, an ontology engineer should also be aware that when building a sound

¹Also referred as *Idioms*.

ontology, dealing with *some complexity* is unavoidable. This is because an ontology is expected to be comprehensive and coherent, not leaving behind important concepts and relations, but also not including unnecessary ones. Otherwise, one might be just overestimating the power of the so-called ontology, in terms of semantic expressivity and interoperability. Taking this into account, we believe that using OPs and OPLs is paramount to help dealing with the aforementioned complexity. And we hope to convince the reader of this important point, by presenting some definitions and examples in the next sections of this chapter.

7.2. What an OPL is and how it is represented

An **Ontology Pattern Language** (OPL) is a network of interconnected DROPs that provides holistic support for solving ontology development problems for a specific domain. An OPL contains a set of interconnected DROPs, plus a modeling workflow guiding on how to use and combine them in a specific order, and suggesting patterns for solving some modeling problems in that domain [6].

The notion of OPL provides a stronger sense of connection between DROPs, expressing different types of relationships among them [6]. For instance, to be applied, a pattern may require the previous application of other patterns (dependency); a larger pattern can be composed of smaller ones; or several patterns may solve the same problem in different ways (variant patterns). Those relationships impose constraints in the order in which patterns can be applied. Thus, an OPL provides explicit guidance on how to reuse and integrate related patterns into a concrete conceptual model of a new domain ontology. In this sense, an OPL is more than a catalogue of patterns. It includes, besides the patterns themselves, a rich description of their relationships, and a process guiding the order to apply them according to the problems to be modeled. OPLs encourage the application of one pattern at a time, following the order prescribed by paths chosen throughout the language. Consequently, by enabling the selective use of DROPs in a flexible way, an OPL releases the ontology engineer from the need to know up-front the whole set of concepts and relations addressed by the OPL. In this way, an OPL also aids managing complexity.

UML activity diagrams can be used for representing OPL process models, since they provide the main constructs for representing process models in general. Figure 7.1 shows an extension of a fragment of the UML meta-model for activity diagrams (version 2.5 [20]) that is adequate for representing OPLs. Modeling elements shown in grey are those extending the UML meta-model. Figure 7.2 shows the graphical notation for the modeling elements used to represent OPLs.

In this extended version of Activity Diagrams, a *Pattern Action* is an action in which a pattern is applied. Pattern Actions are represented as filled round-cornered rectangles. The name of the pattern is used to indicate the pattern to be applied. Pattern actions can be grouped in *Pattern Action Groups*. A Pattern Action Group is used to organize an OPL, aggregating pattern actions and other pattern action groups related to a more specific subject or sub-domain. Pattern Action Groups are represented as hollow round-cornered rectangles with wider borders. A special type of Pattern Action Group is the *Variant Pattern Action*

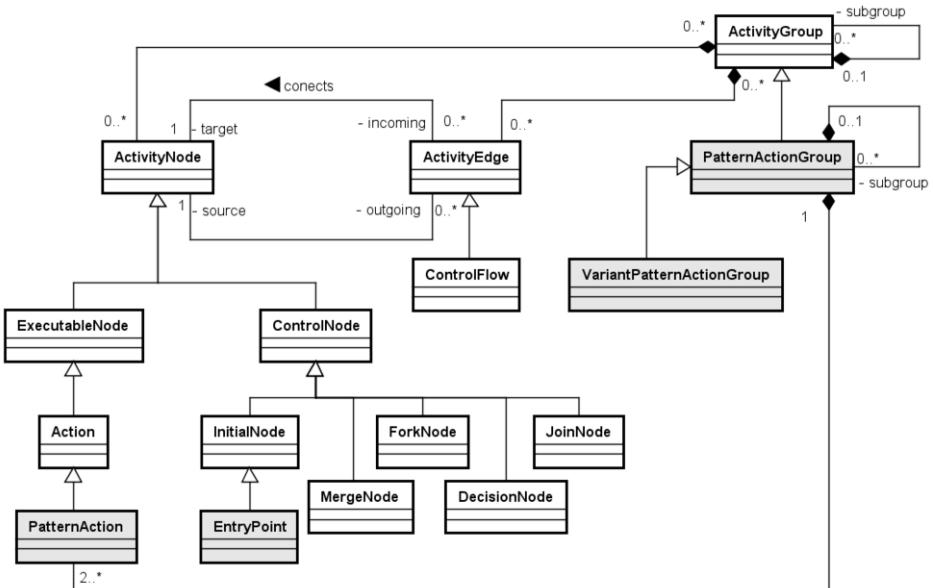


Figure 7.1. An extension of the UML meta-model for representing OPLs.

Group, which aggregates variant pattern actions. Variant patterns groups aggregate patterns solving the same problem but in different and mutually exclusive ways. Thus, from a Variant Pattern Action Group only one of the patterns can be selected and applied. Variant Pattern Action Groups are represented as hollow round-cornered rectangles with wider dashed borders.

Control Flows represent the admissible sequences in which patterns in the language can be applied. A control flow is represented by an arrowed line connecting either: two pattern actions; a pattern action and a (variant) pattern action group; two (variant) pattern action groups. It is important to point out that, in an OPL, control flows indicate admissible paths along the OPL, i.e., they do not represent mandatory paths to be followed. The ontology engineer may decide to stop applying the patterns at any time. However, if he/she wants to proceed, the paths given by the control flows must be respected. When the application of a pattern requires applying the next one, the control flow linking the two corresponding pattern actions is stereotyped as «mandatory».

Besides the main modeling elements for representing pattern actions, groups of pattern actions and control flows between them, in order to allow representing more complex flows than sequential ones, control nodes are used. For representing OPLs, the following control nodes are used: entry point, decision and merge nodes, and fork and join nodes.

Entry Points indicate the patterns in the language that can be used without requiring the previous application of other patterns. There may be multiple entry points in an OPL, indicating different ways of using the OPL. When using an OPL,

Modeling Element	Notation
Pattern Action	
Control Flow	
Pattern Action Group	
Variant Pattern Action Group	
Entry Point	
Decision Node	
Merge Node	
Fork Node	
Join Node	

Figure 7.2. Graphical notation for representing OPLs.

the ontology engineer must choose the entry point that better fits the requirements for the domain ontology being developed. Different from Initial Nodes in UML Activity Diagrams, in an OPL only one entry point is to be selected by the ontology engineer. Entry points (as UML initial nodes) are represented by solid circles.

As in the UML meta-model [20], a *Decision Node* is a control node that selects among alternative outgoing flows, while a *Merge Node* is a control node that brings together multiple flows without synchronization. The notation for both Merge Nodes and Decision Nodes is a diamond-shaped symbol. The difference between them is that a Merge Node must have two or more incoming flows and a single outgoing flow, while a Decision Node must have a single incoming flow and multiple outgoing flows.

A *Fork Node* is a control node that splits a flow into multiple concurrent flows. A Fork Node has exactly one incoming control flow, though it may have multiple outgoing control flows. As discussed above for control flows, these outgoing flows may optionally be followed by the ontology engineer depending on the requirements for the ontology being developed. In other words, a fork node only indicates that the outgoing flows are admissible, but not that they are necessarily mandatory. If some of the outgoing flow is mandatory, the corresponding control flow must be stereotyped with «mandatory». A *Join Node* is a control node that

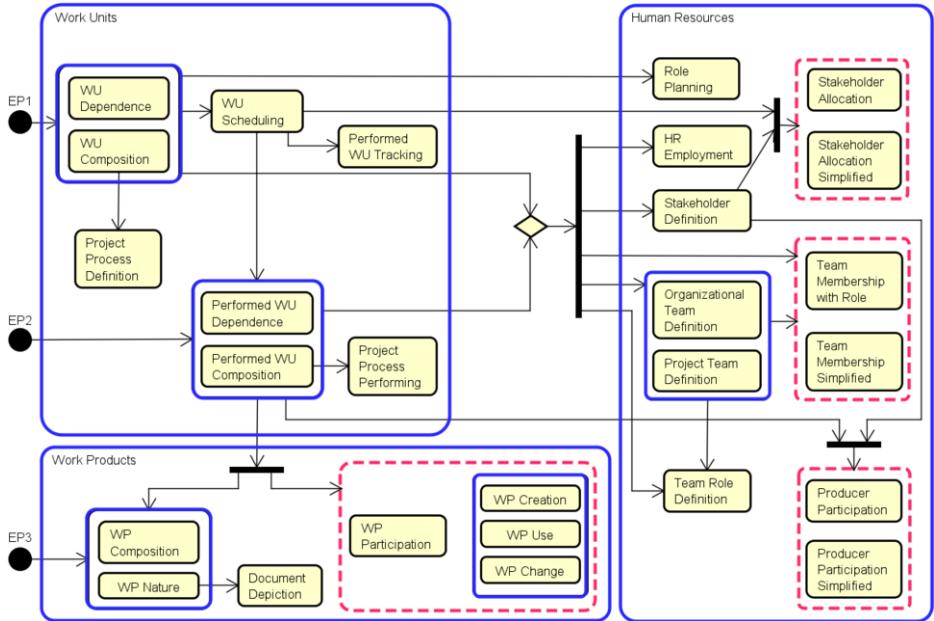


Figure 7.3. ISP-OPL process model.

synchronizes multiple flows. A Join Node shall have exactly one outgoing control flow but may have multiple incoming flows. The notation for both Fork Nodes and Join Nodes is a thick line segment.

Figure 7.3 shows the process model of the ISO-based Software Process OPL (ISP-OPL) [22]. The purpose of ISP-OPL is to establish a common conceptualization about the software process domain, considering ISO Standards devoted to this subject, such as ISO/IEC 24744 [15], ISO/IEC 12207 [16] and ISO/IEC 15504 [14]. The main intended use for ISP-OPL is supporting ISO-Standard harmonization efforts. As Figure 7.3 shows, ISP-OPL is organized in three main groups of patterns: Work Units (WUs), Work Products (WPs) and Human Resources (HRs). ISP-OPL has three entry points. The ontology engineer should choose one of them, depending on the scope of the specific software process ontology being developed. The ontology engineer should choose EP1, when the requirements for the new ontology include the definition and planning of work units; he/she should choose EP2, if the scope of the ontology considers only the execution of work units (performed WUs); EP3 is to be chosen if the ontology engineer aims to model only the structure of work products.

Through entry point EP1, in order to model the structure of WUs, the ontology engineer needs to apply one (or both) of the following patterns: *WU Composition* and *WU Dependence*. These patterns are used to represent work units defined in an endeavor, without planning a time frame for them. The *WU Composition* pattern represents the mereological decomposition of work units, thus, specializing Work Unit into Process, Composite Task and Simple Task. The *WU Dependence* pattern deals with the dependence between work units. The *Project*

Process Definition pattern captures the link between a Process and the Project for which it is defined. The *WU Scheduling* pattern is used to represent the time frame of a scheduled WU, defining its planned start and end dates.

After doing that, the ontology engineer can focus on modeling performed work units, i.e., work units already executed. Performed WUs, as past events, have actual start and end dates. The tracking of performed work units against defined work units is treated by the *Performed WU Tracking* pattern, which relates a Scheduled Work Unit to a Performed Work Unit caused by the former. The group encompassing the patterns *Performed WU Composition* and *Performed WU Dependence* is analogous to the group containing *WU Composition* and *WU Dependence*. Additionally, the Project in the context of which a Process was performed can be modeled with the *Project Process Performing* pattern.

If the requirements for the ontology involve only performed work units, the entry point is EP2, allowing using, in this group, only the patterns *Performed WU Composition*, *Performed WU Dependence* and *Project Process Performing*.

After modeling aspects related to Work Units, the ontology engineer can address human resource related problems by applying the patterns of the Human Resource group. Some patterns of this group are adapted from patterns of the Enterprise OPL (E-OPL) [9] (see Section 7.5.1), such as *Human Resource Employment*, *Organizational Team Definition* and *Project Team Definition*.

The *Human Resource Employment* pattern establishes the employment relation between an Organization and a Person, which assumes the Human Resource role. The *Stakeholder Definition* pattern defines the concept of Stakeholder (someone involved in a Project). The *Organizational Team Definition* and *Project Team Definition* patterns are used to define organizational and project teams, respectively. The *Role Planning* pattern models the roles responsible for performing a defined work unit, while the *Team Role Definition* pattern can be applied to represent the roles a team can play.

In order to represent the membership relation between a team and its members (Persons), the ontology engineer can choose one of the alternative patterns *Team Membership Simplified* and *Team Membership with Role*. Then, one of these two alternative patterns can be used to represent the allocation of stakeholders to a scheduled work unit: *Stakeholder Allocation* and *Stakeholder Allocation Simplified*. Finally, for dealing with the participation of stakeholders in performed work units, the ontology engineer can choose between the alternative patterns *Producer Participation* and *Producer Participation Simplified*.

The last group of patterns constituting ISP-OPL is the group related to Work Products (WPs). This group can be reached from the patterns related to Performed WU, but also through the entry point EP3. This group is to be chosen when the ontology engineer wants to represent only the structure of work products. The *WP Composition* pattern allows modeling the mereological structure of work products. *WP Nature* is related to types of work products (such as Document, Model and Information Item). Once the *WP Nature* pattern has been applied, the *Document Depiction* pattern can be used to model the fact that documents depict other work products.

Once the patterns for work unit execution have already been applied (either through EP1 or EP2), beyond the work product structure, the ontology engi-

neer can also model work products handling. In this case, the *WP Participation* pattern sets the participation of work products in performed work units. In this pattern, the Work Product Participation is modeled as a concept with specializations for creation, change and usage participation. Alternatively, these three types of participations can be modeled only by means of a relation using the patterns *WP Creation*, *WP Change* and *WP Use*.

In addition to the process model, an OPL comprises the patterns themselves. The OPL specification includes the following items for each pattern:

- **Name:** the name of the pattern.
- **Intent:** describes the pattern purpose.
- **Rationale:** describes the rationale underlying the pattern.
- **Competency Questions:** describes the competency questions that the pattern aims to answer.
- **Conceptual Model:** depicts the conceptual model representing the pattern elements.
- **Axiomatization:** presents complementary formal axioms related to the diagrammatic form of the conceptual model. Those axioms typically capture constraints and other aspects of the pattern that cannot be directly represented by the diagrammatic form of the conceptual model.
- **Complementary Patterns:** list other ontology patterns that are related to the pattern being presented, but that are not part of the OPL to which the pattern being described belongs.

As an example, Table 7.1 shows the specification of the *Performed Work Unit Composition* pattern [22].

It is important to highlight that the term “pattern language” was borrowed from Software Engineering (SE), where patterns have been studied and applied for a long time. Thus, we are not actually talking about a language properly speaking. In “pattern language,” the use of the term “language” is, in fact, a misnomer, given that a pattern language does not typically define *per se* a grammar with an explicit associated mapping to a semantic domain. Moreover, although an OPL provides a process describing how to use the patterns to address problems related to a specific domain, an OPL is not a method for building ontologies. Instead, it only deals with reuse in ontology development, and its guidance can be followed by ontology engineers using whatever ontology development method that considers ontology reuse as one of its activities [6].

7.3. Building OPLs from Core Ontologies

For building an OPL for some domain, we need to have a set of interconnected patterns for this domain. A good option to get these patterns is to extract them from ontologies developed for this domain, in particular core ontologies. A core ontology provides a precise definition of the structural knowledge in a specific field that spans across several application domains in that field. Core ontologies are

Performed WU Composition

Name: Performed Work Unit Composition (PWUC)

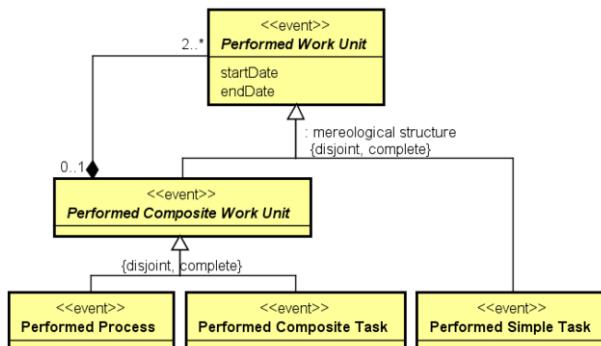
Intent: To represent the composition of *performed work units* in terms of other *performed work units*.

Rationale: *Performed Work Units* can be composed of other *performed work units*. From a mereological point of view, a *performed work unit* is simple, or composed of two or more parts. At the basic level, there are *Performed Simple Tasks* that can compose other *performed work units*, but which are not decomposable. *Performed Composite Tasks*, in turn, are composed of other performed tasks (*composite* or *simple performed tasks*). At the higher level, *Performed Processes* are also composed of performed tasks, but do not compose any other *performed work unit*.

Competency Questions:

- Concerning their mereological structure, what are the possible types of *performed work units*?
- How is a *performed work unit* composed of other *performed work units*?

Conceptual Model



Axiomatization (partial)

A1: $\forall w: \neg \text{partOf}(w, w)$

No individual (and, hence, no *Performed Work Unit*) can be part of itself.

A2: $\forall p: \text{PerformedProcess}(p) \rightarrow \neg \exists w \text{ PerformedWorkUnit}(w) \wedge \text{partOf}(p, w)$

A *Performed Process* cannot be part of any *Performed Work Unit*.

A3: $\forall w_1, w_2: (\text{Event}(w_1) \wedge \text{Event}(w_2) \wedge \text{partOf}(w_2, w_1)) \rightarrow (\text{startDate}(w_2) \geq \text{startDate}(w_1)) \wedge (\text{endDate}(w_2) \leq \text{endDate}(w_1))$

A *Performed Work Unit* that is part of another *Performed Work Unit* should occur within the time interval of the latter.

Table 7.1. The Specification of the Performed WU Composition Pattern.

conceived mainly aiming at reuse, and thus, a pattern-oriented design approach is appropriate for organizing them. By following a pattern-oriented design approach, core ontologies become modular and extensible [24]. Moreover, by providing a network of patterns and rules on how they can be combined, an OPL improves the potential for reuse of a core ontology, by enabling the selective use of parts of the core ontology in a flexible way. This is very important due to pragmatic reasons, since ontology engineers developing specific ontologies for that domain might want to focus on selected aspects of the domain, disregarding others.

With a core ontology in hands, DROPs can be extracted from it through a fragmentation process. To illustrate the extraction of DROPs from core ontologies, consider the case of ISP-OPL. The patterns of ISP-OPL were extracted from a software process core ontology resulting from the ontological analysis of the ISO/IEC 24744 metamodel [15] (Software Engineering Metamodel for Development Methodologies – SEMDM) performed in [21], plus the application of some patterns of the Enterprise OPL (E-OPL) [9]. Figure 7.4 shows the conceptual model of the part of this core ontology dealing with Work Units. This conceptual model is fragmented in the patterns discussed in Section 7.2.

DROP complexity can vary greatly depending on the domain fragment being represented. Sometimes a DROP contains only two related concepts. This is the case of the patterns *WU Dependence*, *WU Scheduling*, *Project Process Definition*, *Performed WU Tracking*, *Performed WU Dependence* and *Project Process Performing*. This fine-grained fragmentation is useful to prevent the ontology engineer from discarding parts of a pattern. In other situations, a DROP can contain a complex combination of concepts and relations, such as in the case of *WU Composition* and *Performed WU Composition*. The *WU Composition* pattern represents the mereological decomposition of Composite Work Units into other Work Units. Composite Work Units are composed of at least two Work Units, and, according to the ISO Standards, they can be of two types: Processes or Composite Tasks. Work Units that are not composed by other Work Units are said to be Simple Tasks.

An important aspect to highlight is that, as pointed out by Scherp et al. [24], a core ontology should be precise. This is achieved by grounding the core ontology in a foundational ontology. Concepts and relations defined in a core ontology should be aligned to the basic categories of a foundational ontology [24]. By doing that, core ontologies incorporate a solid and semantically precise basis.

The ISO-based software process core ontology illustrated here is based on the Unified Foundational Ontology (UFO) [12] [13]. In fact, as previously mentioned, it is the outcome of an ontological analysis of the ISO/IEC 24744 meta-model [15] in terms of UFO. As a consequence, the resulting patterns extracted from this well-founded ontology are themselves well-founded.

7.4. Using OPLs for building Domain Ontologies

In this section, we discuss how to build a domain ontology using an OPL, illustrating this by an example applying ISP-OPL for building a reference ontology for the Requirements Engineering (RE) process.

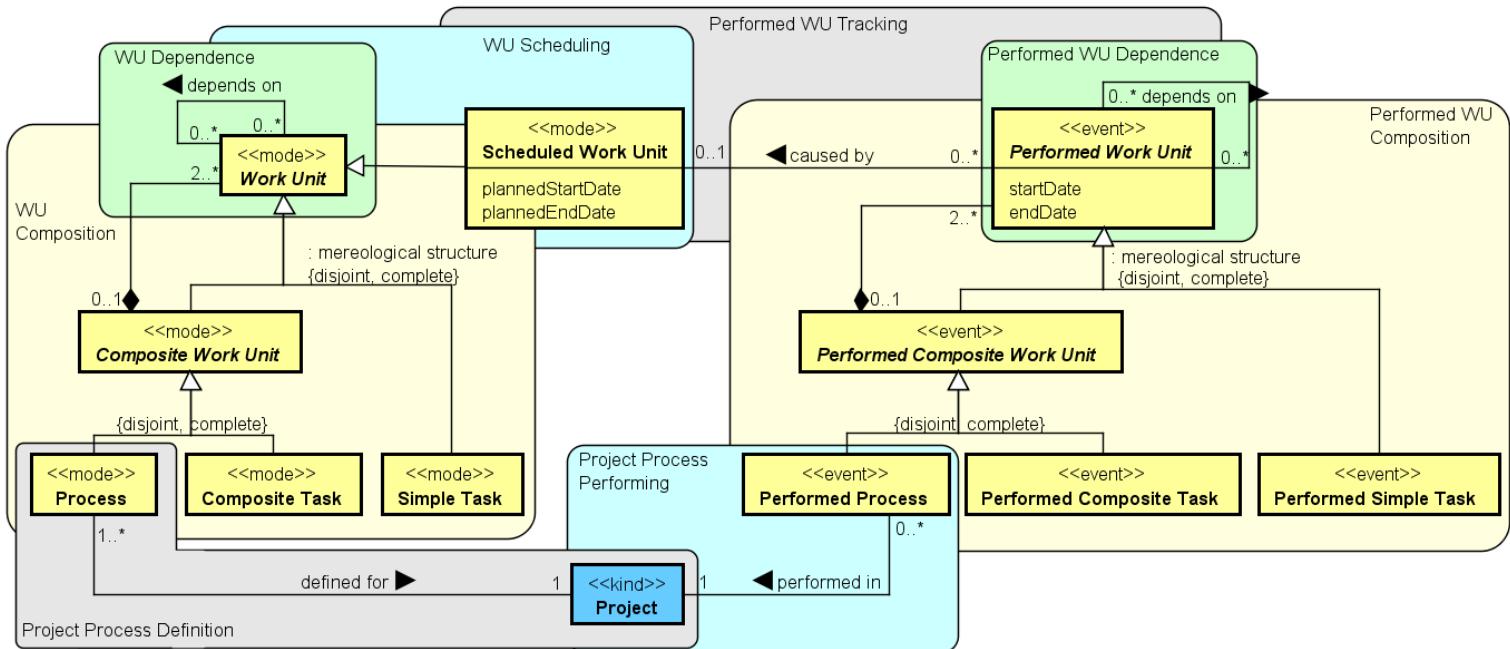


Figure 7.4. Patterns of the Work Unit Group.

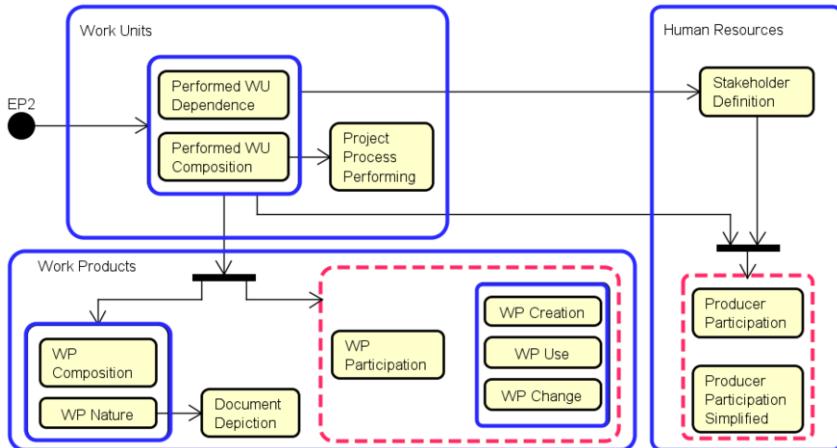


Figure 7.5. ISP-OPL Patterns used and Paths followed (adapted from [22]).

Before using an OPL, the ontology engineer needs to study the OPL process model and its patterns, so that he/she can better decide which entry point is the most suitable for the particular project at hand. Then, he/she has to apply the patterns following the admissible paths through the OPL until there are no more applicable patterns.

The RE Process Ontology presented here was derived from ISP-OPL according to the information extracted from selected ISO SC7 standards [22], namely: ISO/IEC 15288:2008 – System life cycle processes [17], ISO/IEC 12207:2008 – Software life cycle processes [16], and ISO/IEC/IEEE 29148:2011 – Requirements Engineering [18]. These standards define three requirements-related processes: *Stakeholder Requirements Definition*, *System Requirements Analysis*, and *Software Requirements Analysis*. In this chapter we present only the sub-ontology addressing the first process: *Stakeholder Requirements Definition* (Section 6.4.1 in ISO 12207 and ISO 15288, and Section 6.2 in ISO 29148).

We are interested in describing the execution of requirements processes, including the participations of human resources and work products, as it can be typically found in the case of organizations adopting these standards in their projects. Thus, we started using ISP-OPL by the entry point EP2. Figure 7.5 shows the chosen patterns and paths of the ISP-OPL process that we have followed for developing this ontology.

Figure 7.6 presents the *Stakeholder Requirements Definition Process* sub-ontology. On the top, the concepts with colored background are the ones defined as part of the ISP-OPL patterns. On the bottom, the concepts with white background are the specific ones from the RE Process Ontology. Relations in the RE Process Ontology are specializations of the homonymous relations in the OPL. Cardinalities are omitted for the sake of legibility.

The *Stakeholder Requirements Definition* process is decomposed into activities, which, in turn, are decomposed into tasks. Thus, we started with the *Performed WU Composition* pattern, modeling the decomposition of performed

work units. The ***Stakeholder Requirements Definition Process*** is a subtype of **Performed Process**. This specialized process is composed of five work units: ***Stakeholder Identification***, ***Requirements Identification***, ***Requirements Evaluation***, ***Requirements Agreement*** and ***Requirements Recording***. The first and fourth work units are **Performed Simple Tasks**, and the others are **Performed Composite Tasks**, which are themselves decomposed into simple tasks as shown in Figure 7.6.

Another pattern considered useful here is ***Performed WU Dependence***, which defines dependencies between work units. Although the selected standards do not explicitly set dependencies between tasks, some of them can easily be inferred from the nature of work units and work products handled, as well as by considering the RE literature. Accordingly, we applied the ***Performed WU Dependence*** pattern to establish dependencies between the work units as shown in Figure 7.6. Still regarding work units, the last pattern applied was ***Project Process Performing***, establishing the connection between the **Performed Process** and the **Project** wherein it is performed.

Once work units have been addressed, we could address human resources related problems. Due to the general nature of the standards, few information is given about human resources participating in work units. Thus, we have modeled only the stakeholder definition and its relation to work units. The first pattern applied was ***Stakeholder Definition***, in order to establish the types of stakeholders to be considered. We considered only two types of stakeholders: ***System Analyst***, and ***Requirements Stakeholder***. Both of which are types of **Person Stakeholder** involved in a **Project**. Aiming at representing the participation of stakeholders in work units, the ***Producer Participation Simplified*** pattern was used, specializing **Stakeholders** as **Producers** who participate in **Performed Work Units**.

The other path of ISP-OPL we followed is through the use of the work products patterns. Once there are different types of work products, it is useful to distinguish between them by applying the ***WP Nature*** pattern. Two subtypes of **Work Product** were considered: **Information Item** and **Document**. In the context of the ***Stakeholder Requirements Definition Process***, we identified the following subtypes of **Information Item**: ***Requirement*** (specialized into ***Stakeholder Requirement***), ***Stakeholder List***, ***Stakeholder Agreement***, and ***Traceability Record***. Moreover, two sub-types of **Document** were considered: ***Requirements Evaluation Doc***, and ***Stakeholder Requirements Specification***. The ***Stakeholder Requirements Specification*** is the main result of this process and aggregates the ***Stakeholder List*** and the set of ***Stakeholder Requirements***. Thus, using the ***WP Composition*** pattern, we establish ***Stakeholder Requirements Specification*** as a **Composite Work Product**, composed of ***Stakeholder Requirements*** and ***Stakeholder List*** (both of which are types of **Simple Work Product**). Additionally, by applying the ***Document Depiction*** pattern, we represented a ***Stakeholder Requirements Specification***, which (as a document) represents the ***Stakeholder Requirements***.

Finally, by using the patterns ***WP Creation***, ***WP Use*** and ***WP Change***, we established the relations of creation, usage and change between the work units of the ***Stakeholder Requirements Definition Process*** and their corresponding

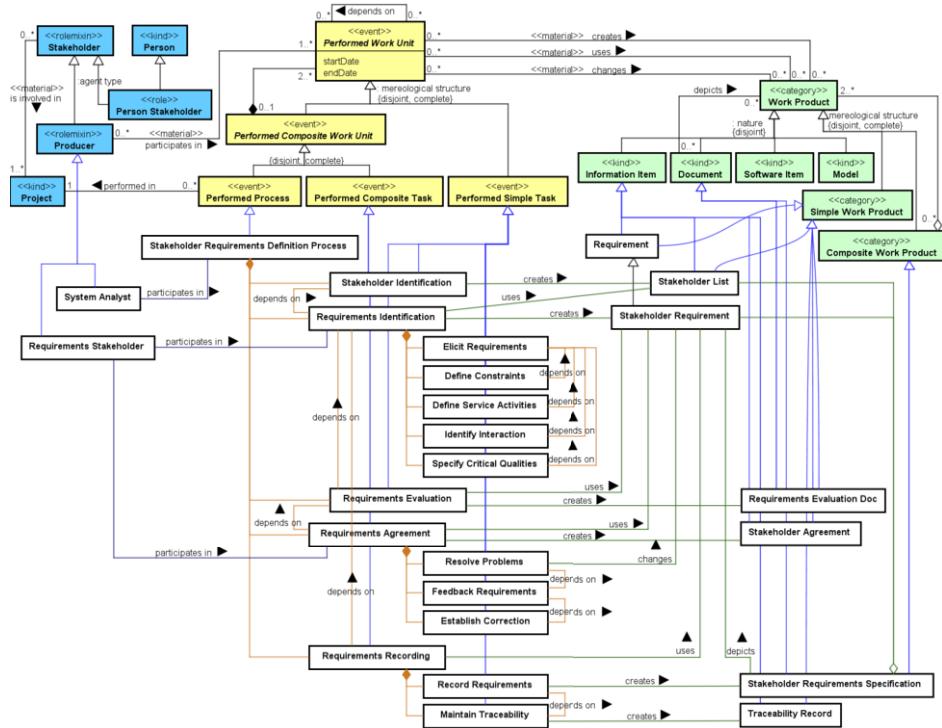


Figure 7.6. The Stakeholder Requirements Definition Process sub-ontology [22].

work products.

It is important to point out that reuse is not limited to the conceptual models, as discussed above. As Table 7.1 shows, the patterns' specification includes also competency questions (CQs) and axioms. Typically, these elements are also reused when a pattern in the OPL is selected. Once a DROP is chosen, its concepts and relations become part of the domain ontology (as Figure 7.6 shows), where they can be further extended. For CQs, a very similar approach holds: once a DROP is chosen, its CQs can be extended for the domain ontology. For example, the *Performed WU Composition* pattern (see Table 7.1) has the following CQs: (i) *Concerning their mereological structure, what are the possible types of performed work units?*; and (ii) *How is a performed work unit composed of other performed work units?* When this pattern was applied to the Software RE process, the following (extended) specific CQs were created: (i) *What are the possible types of performed work units in the RE process?*; and (ii) *How is the RE process decomposed?* This reuse helps with the definition of CQs, improving the productivity of the ontology engineering process [23]. The same applies to the case of reusing general axioms.

7.5. Existing OPLs

The use of OPLs is a recent initiative. There are still only few works defining OPLs, among them the following: Software Process OPL (SP-OPL) [6], ISO-based Software Process OPL (ISP-OPL) [22], Enterprise OPL (E-OPL) [9], Measurement OPL (M-OPL) [2], and Service OPL (S-OPL) [8]. ISP-OPL was presented in the previous section. SP-OPL is related to the same domain (software processes), but it is more general than ISP-OPL since it is not devoted to ISO Standards. Thus, in this section, we briefly present the other three OPLs. Further information on these OPLs can be found at <http://nemo.inf.ufes.br/OPL>.

7.5.1. Enterprise OPL

The Enterprise Ontology Pattern Language (E-OPL) [9] aims at providing patterns for enterprise ontology modeling. It is composed of 22 patterns addressing five aspects common to several enterprises:

- **Organization Arrangement** (4 patterns): includes patterns related to how multi-organizations are organized in terms of other organizations (*Multi Organization Arrangement* pattern), how a complex organization is structured in terms of organizational units (*Complex Organization Arrangement* pattern), as well as how complex organizational units are structured in terms of other organizational units (*Complex Organizational Unit Arrangement* pattern). Finally, if organizations to be modeled are simple (not composed of other organizations or organizational units) or addressing organizations' structure is out of the scope of the domain ontology being developed, then there is the *Simple Organization Arrangement* pattern;
- **Team Definition** (3 patterns): deals with defining teams for projects (*Project Team Definition* pattern), organizations (*Organizational Team Definition* pattern) and organizational units (*Organizational Unit Team Definition* pattern);
- **Institutional Roles** (4 patterns): addresses the representation of roles and positions to be played by enterprise employees. This group contains the following patterns: the *Organizational Positions* pattern deals with positions defined in an organization; the *Organizational Roles* pattern addresses roles defined in an organization; the *Organizational Unit Roles* and *Team Roles* patterns concern informal roles defined by an organizational unit or a team, respectively;
- **Institutional Goals** (3 patterns): deals with institutional agents' goals, and there are three patterns available: *Organizational Goals*, *Organizational Unit Goals*, and *Team Goals* patterns;
- **Human Resource Management** (8 patterns): treats the following human resource relations in an enterprise: employment, allotment to an organizational unit, team allocation, and position occupation. For each one of these relations, two variant patterns are defined, one explicitly including a

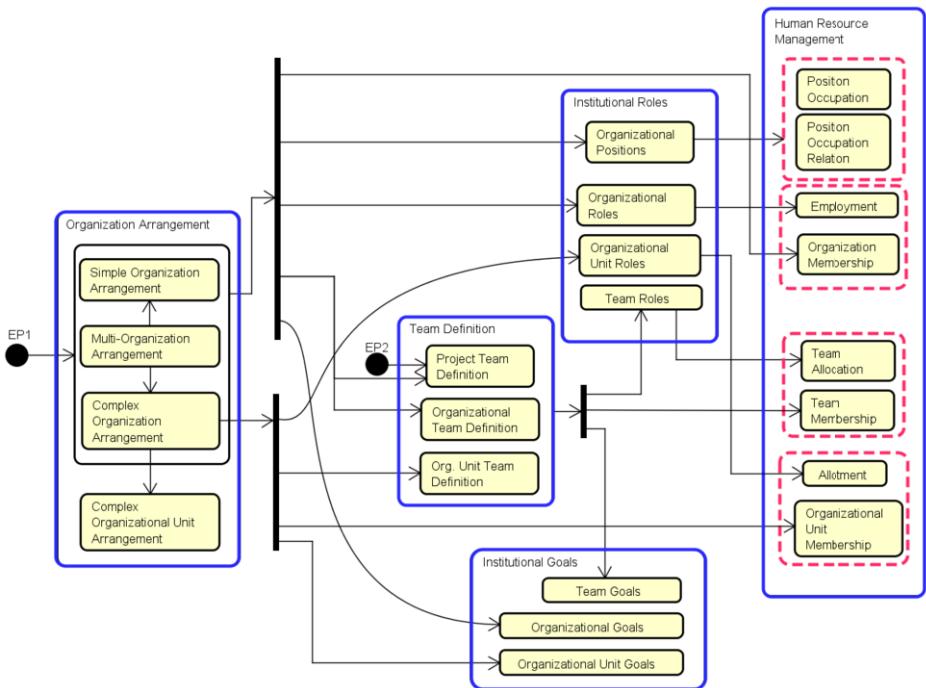


Figure 7.7. E-OPL Process Model (adapted from [9]).

concept reifying the material relation (a relator), and another disregarding the relator and considering only the material relation.

Figure 7.7 shows the E-OPL process model. As this figure shows, E-OPL has two entry points. The ontology engineer should choose one of them, depending on the scope of the specific enterprise ontology being developed. When the requirements for the new enterprise ontology being developed include only problems related to the definition of project teams, the starting point is EP2. Otherwise, the starting point is EP1. In this case (EP1), first the ontology engineer should address problems related to how an organization is structured. One of the three following patterns has to be selected: *Simple Organization Arrangement* pattern, *Complex Organizational Unit Arrangement* pattern or *Multi-Organization Arrangement* pattern.

Simple Organization Arrangement pattern should be selected as the first pattern if the ontology engineer needs to represent only very simple standalone organizations, which are neither composed of other organizations nor composed of organizational units. The *Complex Organization Arrangement* pattern should be selected as the first pattern if the ontology engineer needs to represent only complex standalone organizations, which are not composed of other organizations, but which are composed of organizational units. When the *Complex Organization Arrangement* pattern is selected, the *Complex Organizational Unit Arrangement* pattern can be used in the sequel, if there is a need to represent complex orga-

nizational units, which are composed of other organizational units. Finally, the *Multi-Organization Arrangement* pattern should be selected as the first pattern if the ontology engineer needs to represent organizations that are composed of other organizations. When the *Multi-Organization Arrangement* pattern is used, the ontology engineer can also use the *Simple Organization Arrangement* and *Complex Organization Arrangement* patterns to address the organizational structure of the standalone organizations that compose a multi-organization.

Once problems related to the organization arrangement are addressed, the ontology engineer can treat problems related to the definition of organizational teams, goals and roles, and some problems related to human resource management.

Concerning team definition, three types of teams are considered: organizational teams, organizational unit teams and project teams. The *Project Team Definition* pattern deals with teams that are defined with the specific purpose of performing a project. For this reason, this pattern does not require the problems related to organizational arrangement to be addressed prior to its use. For this reason, it takes EP2 to be its entry-point in E-OPL. The *Organizational Team Definition* and the *Organizational Unit Team Definition* patterns deal respectively with organizational and organizational unit teams.

Regarding goals, in a general view, in E-OPL, Institutional Agents (a generalization for Organizations, Organizational Units and Teams) may define Institutional Goals, and three patterns are available: *Organizational Goals*, *Organizational Unit Goals*, and *Team Goals*.

Concerning roles, Institutional Agents (Organizations, Organizational Units and Teams) may define Institutional Roles. Like the TOVE Ontology [10], E-OPL considers two main types of Institutional Roles: Positions and Human Resource Roles. A Position represents some formal position in the organization, such as “president”, “sales manager”, etc. A Human Resource Role defines a prototypical function of a person in the scope of an Institutional Agent, such as “engineer” or “system analyst”. Moreover, E-OPL distinguishes between formal and informal roles. Formal Human Resource Roles are those recognized by the whole organization and its environment (partners and society in general). Informal Human Resource Roles are those recognized only in the scope of the corresponding institutional agent. Team Roles and Organizational Unit Roles are types of informal roles, recognized, respectively, by a Team and by an Organizational Unit. Organizational Roles can be formal or informal. Formal Organizational Roles are those considered when employments are created. Each employment is made for a specific formal role. On the other hand, a particular person, in the same employment, can assume several informal roles.

In order to deal with institutional roles, four patterns were defined. The *Organizational Positions* pattern deals with positions defined in an organization. The *Organizational Roles* pattern addresses both formal and informal roles defined in an organization. The *Organizational Unit Roles* pattern and *Team Roles* pattern address informal roles defined by an organizational unit or a team, respectively.

Finally, problems related to human resource management can be addressed. In E-OPL there are eight patterns treating four material relations that involve human resources, namely: employment, allotment, team allocation, and occupa-

tion.

According to UFO, material relations are relations that have material structure on their own. The relata of a material relation are mediated by individuals that are called relators. Relators are complex objectified relational properties [12] [11]. From a pragmatic point of view, depending on the requirements of the enterprise ontology being developed, the ontology engineer may be interested in showing only the material relation instead of showing the whole model, including also the relator and the mediation relations. Thus, each one of these material relations is addressed by two patterns: one explicitly including a concept representing the relator (*Position Occupation*, *Employment*, *Team Allocation* and *Allotment* patterns), and another disregarding the relator and considering only the material relation (*Position Occupation Relation*, *Organization Membership*, *Team Membership* and *Organizational Unit Membership* patterns, respectively). The patterns considering the relators are more complete. They allow capturing information about the relator (for instance, start date and end date of a position occupation, employment, team allocation, or allotment). However, if for a given context, such information is considered irrelevant, then the ontology engineer can choose simpler patterns, which disregard the respective relators, capturing only the fact that human resources are members of organizations, organizational units, and teams, or that they are capable holding positions.

For details regarding E-OPL and its patterns, see [9].

7.5.2. Service OPL

The Service Ontology Pattern Language (S-OPL) [8] aims at providing a network of interconnected ontology modeling patterns covering the core conceptualization of services. S-OPL builds on UFO-S, a commitment-based core ontology for services [19]. The S-OPL patterns support modeling types of customers and providers, as well as the main service life-cycle phases, namely: service offering, service negotiation/agreement, and service delivery. S-OPL is composed of 48 patterns, grouped in 4 groups, namely:

- **Service Offering** (4 patterns): includes patterns to deal with service offering (*SOffering* pattern), description of a service offering (*SODescription* pattern) as well as commitments and claims of the provider (*SOCommitments* and *SOClaims* patterns);
- **Service Negotiation and Agreement** (9 patterns): includes patterns that deal with service negotiation and agreement (*SNegotiation*, *SAGreement*, *SNegAgree* and *SOfferAgree* patterns), description of a service agreement (*SADescription* pattern) as well as commitments and claims of both the hired provider and the service customer (*HPCCommitments*, *HPCClaims*, *SCCommitments* and *SCClaims* patterns);
- **Service Delivery** (7 patterns): includes patterns to deal with the service delivery (*SDelivery* pattern), the actions performed by both the hired provider and the service customer to deliver the service (*HPActions*, *SCActions* and *Interactions* patterns), as well as the motivation to perform these

actions (*HActionMotivation*, *SCActionMotivation* and *InteractionMotivation* patterns);

- **Service Provider and Customer** (28 patterns): deals with defining the types of agents (Person, Organization or Organizational Unit) that can act as Service Provider, Target Customer, Hired Provider and Service Customer.

Figure 7.8 shows the S-OPL process model. As this figure shows, S-OPL has two entry points (EP1 and EP2). The ontology engineer should choose one of them depending on the scope of the specific domain service ontology being developed. When the requirements for the ontology include describing the service offering, then the starting point is EP1. Otherwise, the starting point is EP2.

In the case EP1 is chosen, the ontology engineer should use first the *SOferring* pattern for modeling the service offering itself. Next, he/she must follow the mandatory path: the one that leads to the *Service Provider and Customer* group, which addresses the issue of modeling which types of providers and target customers are involved in the offering.

Providers and target customers can be people, organizations or organizational units. Therefore, the ontology engineer must select one of the patterns of the *Provider* sub-group, and one of the patterns of the *Target Customer* sub-group. These variant patterns are important, because depending on the nature of the service being modeled, only certain types of customers and providers are admissible. For instance, the passport issuing service is offered only to people. The car rental service, in turn, is offered to people, organizational units, and organizations. Thus, each pattern in the *Service Provider and Customer* group offers a different option for the ontology engineer to precisely decide what kinds of entities can play the roles of provider and customer in the service domain being modeled. For instance, in the *Target Customer* sub-group of variant patterns, *P-TCustomer* should be used when only people can play this role. *O-TCustomer* should be used when only organizations can play this role. *OU-TCustomer* should be used when exclusively organizational units can play this role. *O-OU-TCustomer* should be used when both organizations and organizational units can play this role. *P-O-TCustomer* should be used when both persons and organizations can play this role. *P-OU-TCustomer* should be used when both persons and organizational units can play this role. Finally, *P-O-OU-TCustomer* should be used when any of these kinds of entities can play this role.

Besides mandatorily modeling the types of providers and target customers, the ontology engineer can follow the several paths coming out of the fork node. Thus, he/she can use the patterns *SOCclaims* and *SOCcommitments*, in the cases in which he/she is interested in modeling offering claims and commitments, respectively. In addition, the ontology engineer can also choose the *SODescription* pattern, in case he/she is interested in describing the offering by means of a service offering description.

Once the service offering is modeled, the ontology engineer is able to address problems related to service negotiation and agreement. However, service offering may be out of the scope of the ontology. In this case, EP2 should be the entry point in the S-OPL process.

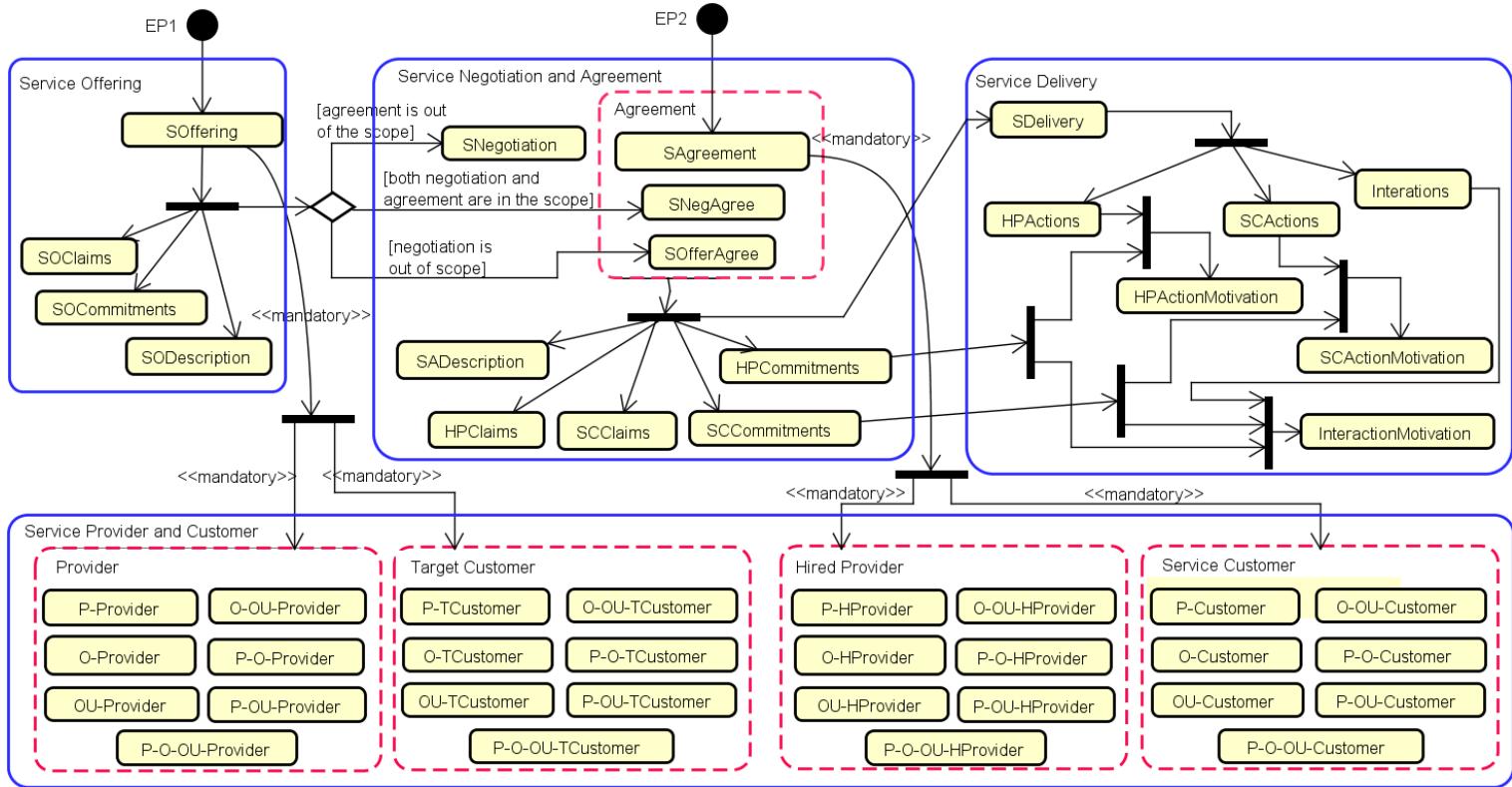


Figure 7.8. S-OPL Process Model (adapted from [8]).

If the ontology engineer has already modeled the service offering, he/she must decide first whether he/she needs to represent service negotiation and/or service agreement. If he/she wants to model only the service negotiation, without modeling the agreement that could result from it (i.e., the agreement is out of scope), he/she should use the *SNegotiation* pattern. If he/she needs to model both the negotiation and the agreement, then he/she should use the *SNegAgree* pattern. Finally, if negotiation is out of the ontology scope, then he/she should use the *SOfferAgree* pattern, which represents an agreement in conformance to an offering.

If EP2 is the entry point in the process, the first pattern to be used is *SAgreement*. In the sequel, the ontology engineer must select one of the patterns of the *Hired Provider* sub-group and one of the patterns of the *Service Customer* sub-group, in order to model the possible types of hired provider and service customer, respectively. The patterns in the *Hired Provider* and *Service Provider* sub-groups are analogous to the ones in the *Provider* and *Target Customer* sub-groups respectively. Note that defining the types of hired providers and service customers is necessary only if the chosen entry point is EP2, since in cases in which the entry point in the process is EP1, the types of providers and target customers would already have been modeled at that point.

Once the agreement is modeled, the following patterns can be optionally used: *HPCCommitments* and *HPCClaims*, depending whether the ontology engineer is interested in modeling the hired provider commitments and claims, respectively; *SCCCommitments* and *SCClaims*, depending on whether he/she is interested in modeling service customer commitments and claims, respectively; and *SADescription*, in case he/she is interested in describing the service agreement by means of a description.

After modeling the agreement, the ontology engineer can model the service delivery. In this group, the first pattern to be used is *SDelivery*. In the sequel, if he/she wants to model the actions involved in a delivery, the following patterns must be applied: *HPActions*, for modeling actions performed by the hired provider; *SCActions*, for modeling actions performed by the service customer; and *Interactions*, for modeling actions performed by both the hired provider and service customer, in conjunction. After that, he/she can model the relationships between the actions and the commitments that motivate them. This can be done by using the following patterns: *HPActionMotivation*, *SCActionMotivation* and *InteractionMotivation*. Since these patterns establish links between commitments and actions, they require the patterns related to the former to be used prior to the patterns related to the latter.

For details regarding S-OPL and its patterns, see [8].

7.5.3. Measurement OPL

The Measurement Ontology Pattern Language (M-OPL) [2] aims at providing a network of ontology modeling patterns addressing the core conceptualization about measurement. It is composed of 19 patterns, organized in 6 groups, namely:

- **Measurable Entities** (2 patterns): includes patterns related to the entities and their properties that can be measured. There are two pat-

terns: *Measurable Entity*, which deals with entities that can be measured, their types and their measurable properties; and *Measurable Element Type*, which is related to the types of measurable properties;

- **Measures** (2 patterns): includes patterns to deal with defining measures (*Measure*) and classifying them according to their dependence on others measures (*Measure Type*);
- **Measurement Units & Scales** (3 patterns): concerns the scales associated to measures and the measurement units used to partition scales. This group includes three patterns: *Measure Scale Type*, which addresses types of scales; *Measure Unit*, dealing with measurement units; and *Measure Scale*, which is related to measure scales;
- **Measurement Procedures** (5 patterns): addresses procedures needed to collect data for measures. Includes five patterns: *Measurement Procedure* deals with the data collection procedure; *Measurement Procedure Type* concerns types of measurement procedures; *Measurement Procedure for Base Measure*, which addresses measurement procedures established w.r.t. base measures; *Measurement Procedure for Derived Measure*, which regards measurement procedures established to derived measures; and *Measurement Formula*, which deals with formulas used to obtain data for derived measures;
- **Measurement Planning** (5 patterns): treats the goals that drive measurement as well as the measures used to verify the achievement of goals. This group includes five patterns, namely: *Measurement Goal*, which deals with measurement goals and information needs to be satisfied by measurement; *Measurement Goal Type*, which is related to measurement goal decomposition; *Measurement Planning Item*, which addresses the basic planning for measurement, including the measurement goal to be monitored, the information need to be met and the measure to be used; *Measurement Planning Item Procedure*, which models measurement planning items adding the measurement procedure to be used; and *Indicator*, which is about identifying measures that act as indicators to goal monitoring;
- **Measurement & Analysis** (2 patterns): includes patterns related to the topics of data collection (*Measurement*) and data analysis (*Measurement Analysis*).

Figure 7.9 shows the M-OPL process model. As this figure shows, M-OPL has only one entry point: EP1.

The first pattern to be used by the ontology engineer is *Measurable Entity*, which models entities that can be measured, their types and properties (i.e., measurable elements). After using this pattern, two patterns are applicable: *Measurable Element Type*, to distinguish direct from indirect measurable elements, and *Measures*, to model measures and relate them to measurable elements and measurable entity types. After the *Measure* pattern, it is possible to apply the *Measure Type* pattern, in order to model the distinction between base and derived measures. However, as shown in Figure 7.9, *Measure Type* can only be used after applying the *Measurable Element Type*, since the measure type is defined based

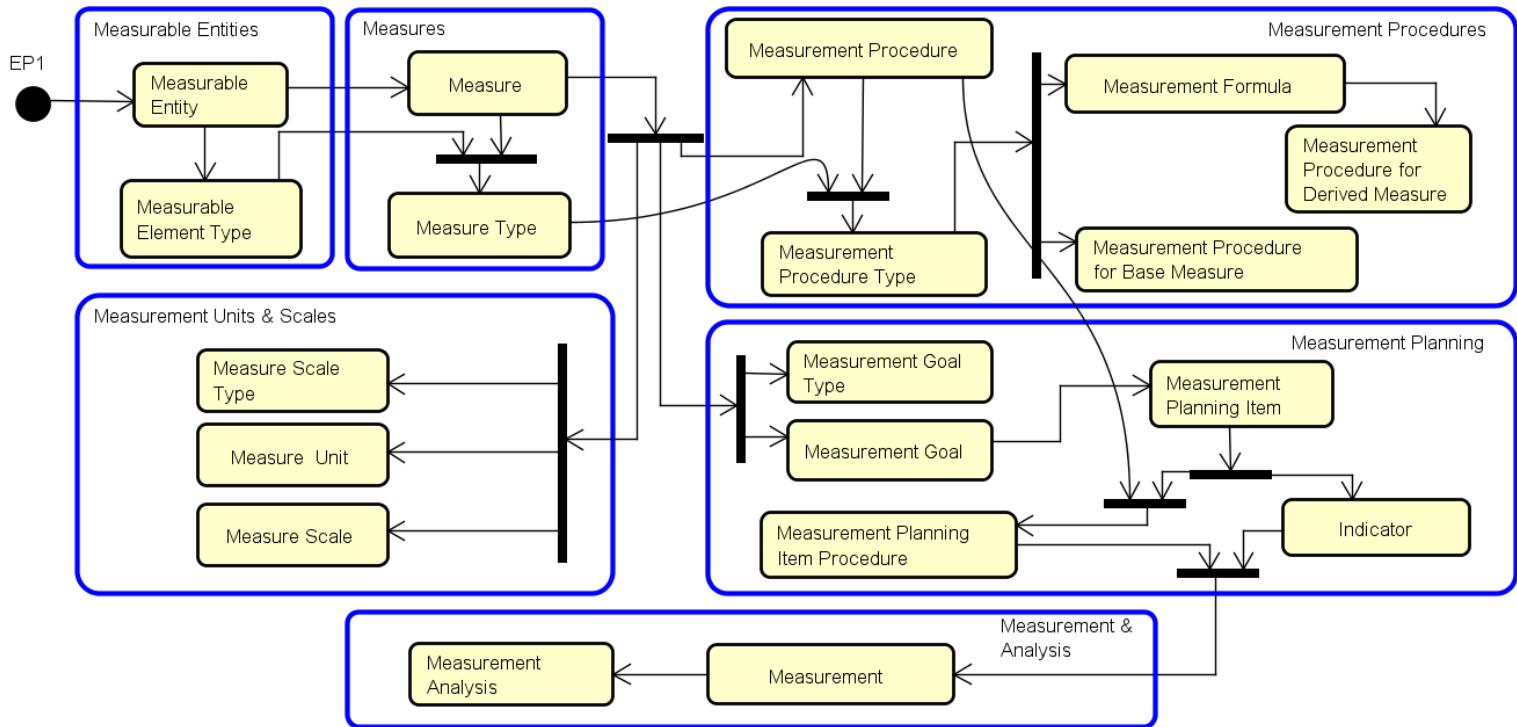


Figure 7.9. M-OPL Process Model (adapted from [2]).

on the type of the measurable element qualified by the measure.

From the *Measure* pattern, there are still three possible paths to follow. The first one goes to the *Measurement Units & Scales* group. In this group, if types of scales (e.g., interval, ordinal and rational) are relevant to the specific measurement domain ontology being developed, the ontology engineer must use the *Measure Scale Type* pattern. If the ontology engineer wants to address measurement units, he/she should use the *Measure Unit* pattern. If measure scales are relevant to the ontology being developed, the *Measure Scale* pattern is to be used.

The second path from the *Measure* pattern goes to the *Measurement Procedures* group. *Measurement Procedure* should be used to model the procedures that guide data collection for measures. If the ontology engineer needs to address different types of measurement procedures, depending on the type of measures, he/she should use the *Measurement Procedure Type*. In this case, besides *Measure*, *Measure Type* should also be used in advance. Measurement procedures for base measures and derived measures are addressed respectively by *Measurement Procedure for Base Measure* and *Measurement Procedure for Derived Measure*. To model measurement procedures for derived measures, the engineer must first use the *Measurement Formula* pattern, which addresses measurement formulas used to calculate derived measures.

The third path from the *Measure* pattern goes to the *Measurement Planning* group. The ontology engineer should use the *Measurement Goal Type* pattern only if representing the decomposition of measurement goal is relevant. In order to address measurement goals and the information needs derived from them, the ontology engineer should use the *Measurement Goal* pattern. Then, if the ontology engineer wants to model measurement planning, addressing measurement goals, information needs and their relation with measures, he/she needs to use the *Measurement Planning Item* pattern. If it is relevant to indicate the measurement procedure in the measurement planning, then, he/she should also use *Measurement Planning Item Procedure*. It is important to notice that, in this case, as shown in Figure 7.9, the *Measurement Procedure* pattern must be used in advance. If the ontology engineer needs to model the relationship between measurement goals and the measures used to indicate their achievement, the *Indicator* pattern is to be used.

Finally, once the measurement planning is addressed, it is possible to model issues related to data collection and analysis. For dealing with data collection aspects, the ontology engineer should use the *Measurement* pattern. The *Measurement Analysis* pattern, in turn, should be used for the ontology engineer to model aspects related to data analysis.

For details regarding M-OPL and its patterns, see [2].

7.6. Final Remarks

This book deals with a range of themes related to ODPs, such as methodologies to use ontology patterns, ontology coding patterns, axiomatization of ODPs, among others. It is important to highlight that although the word “language” in Ontology Pattern Language (OPL) may be misleading, the work presented in

this chapter actually concerns the support for domain ontology development. In this sense, OPLs provide guidelines for the development of the conceptual model of the ontology. Such conceptual model may eventually be implemented using a knowledge representation language, such as OWL, RDF(S), F-Logic, or others, a theme which falls outside the scope this chapter. In any case, the conceptual model representing the ontology has a value in itself, i.e., not only as an essential reference model for ontology implementation, but also supporting important tasks such as meaning negotiation and interoperability.

Summarizing the content of this chapter, we here discussed the notion of Ontology Pattern Languages (OPLs). Moreover, we exemplified how an ontology may be built with the support of an OPL. Finally, we presented existing OPLs on the domains of Software Process, Enterprises, Services and Measurement. The use of OPLs can guide the ontology engineer on selecting specific ontology patterns, depending on the problem being modeled. This may lead to gains in productivity, as well as to improvement in the quality of the resulting ontologies.

The pattern languages presented in this chapter must be seen as work in progress. In fact, OPLs in general should be expected to evolve as a result of new experiences gained throughout their application.

To summarize, an OPL provides concrete guidance, taking into consideration the following questions: (i) What are the main problems to be solved in the targeted domain? (ii) In which order should these problems be addressed? (iii) What alternatives are there to solve such problem? (iv) How should the dependencies among the problems be addressed? (v) How to solve each individual problem effectively, also considering the other problems related to it? In an OPL, the responses to these questions come as a network of ontology patterns that can be combined to create a domain ontology, and a process that guides the consistent application of such ontology patterns.

Completeness and maturity are paramount qualities of a good OPL. Moreover, we claim that OPLs must present some characteristics generally pointed as being present in “beautiful ontologies” [4]: satisfy relevant requirements, have a good coverage of the targeted domain, be often easily applicable in some context, be structurally well designed (either formally or according to desirable patterns), and their domains should introduce constraints that lead to modeling solutions that are non-trivial.

Bibliography

- [1] C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language: towns, buildings, construction*, volume 2. Oxford University Press, 1977.
- [2] M. Barcellos, R. Falbo, and V. Frauches. Towards a measurement ontology pattern language. In *ONTO.COM/ODISE@FOIS*, 2014.
- [3] F. Buschmann, K. Henney, and D. Schimdt. *Pattern-oriented Software Architecture: On Patterns and Pattern Language*, volume 5. John Wiley & sons, 2007.
- [4] M. d’Aquin and A. Gangemi. Is there beauty in ontologies? *Applied Ontology*, 6(3):165–175, 2011.

- [5] P. Deutsch. Models and patterns. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, 2004.
- [6] R. Falbo, M. Barcellos, J. C. Nardi, and G. Guizzardi. Organizing ontology design patterns as ontology pattern languages. In *The Semantic Web: Semantics and Big Data*, pages 61–75. Springer, 2013.
- [7] R. Falbo, G. Guizzardi, A. Gangemi, and V. Presutti. Ontology patterns: clarifying concepts and terminology. In *Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns- Volume 1188*, pages 14–26. CEUR-WS.org, 2013.
- [8] R. Falbo, G. K. Quirino, J. C. Nardi, M. Barcellos, G. Guizzardi, N. Guarino, A. Longo, and B. Livieri. An ontology pattern language for service modeling. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016.
- [9] R. Falbo, F. Ruy, G. Guizzardi, M. Barcellos, and J. P. A. Almeida. Towards an enterprise ontology pattern language. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 323–330. ACM, 2014.
- [10] M. S. Fox, M. Barbuceanu, M. Grunninger, and J. Lin. An organization ontology for enterprise modelling, 1997.
- [11] N. Guarino and G. Guizzardi. We need to discuss the relationship: Revisiting relationships as modeling constructs. In *Advanced Information Systems Engineering*, pages 279–294. Springer, 2015.
- [12] G. Guizzardi. *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology, 2005.
- [13] G. Guizzardi, R. Falbo, and R. Guizzardi. Grounding software domain ontologies in the unified foundational ontology (UFO): The case of the ODE software process ontology. In *CIBSE*, pages 127–140, 2008.
- [14] ISO/IEC. ISO/IEC 15504. *Information Technology as Process Assessment. Part 1: Concepts and Vocabulary*, 2004.
- [15] ISO/IEC. ISO/IEC 24744. *Software Engineering - Metamodel for Development Methodologies*, 2007.
- [16] ISO/IEC. ISO/IEC 12207. *Systems and Software Engineering - Software Life Cycle Processes*, 2008.
- [17] ISO/IEC. ISO/IEC 15288. *Systems and Software Engineering - System Life Cycle Processes*, 2008.
- [18] ISO/IEC/IEEE. ISO/IEC/ieee 29148. *Systems and software engineering - Life cycle processes - Requirements engineering*, 2008.
- [19] J. C. Nardi, R. Falbo, J. A. P. Almeida, G. Guizzardi, L. F. Pires, M. J. van Sinderen, N. Guarino, and C. Fonseca. A commitment-based reference ontology for services. *Information systems*, 54:263–288, 2015.
- [20] Object Management Group (OMG). Unified modeling language (UML), version 2.5.
- [21] F. Ruy, R. Falbo, M. Barcellos, and G. Guizzardi. An ontological analysis of the ISO/IEC 24744 metamodel. In *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS'2014)*, pages 330–343, 2014.
- [22] F. Ruy, R. Falbo, M. Barcellos, G. Guizzardi, and G. Quirino. An ISO-based software process ontology pattern language and its application for harmoniz-

- ing standards. *ACM SIGAPP Applied Computing Review*, 15(2):27–40, 2015.
- [23] F. Ruy, C. Reginato, V. Santos, R. Falbo, and G. Guizzardi. Ontology engineering by combining ontology patterns. In *Proceeding of the 34th International Conference on Conceptual Modeling (ER'15)*, pages 173–186. Springer, 2015.
 - [24] A. Scherp, C. Saathoff, T. Franz, and S. Staab. Designing core ontologies. *Applied Ontology*, 6(3):177–221, 2011.
 - [25] D. C. Schmidt, M. Stal, H. Rohnert, F. Buschmann, and J. Wiley. *Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley, 2000.

This page intentionally left blank

Chapter 8

Anti-patterns in Ontology-driven Conceptual Modeling: The Case of Role Modeling in OntoUML

Tiago Prince Sales, Department of Information Engineering and Computer Science, University of Trento, Italy

Giancarlo Guizzardi, Ontology and Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo, Brazil

8.1. Introduction

Given the increasing complexity of ontology-driven conceptual modeling and ontology engineering, there is an urging need for developing a new generation of complexity management tools for these disciplines [12]. These include a number of methodological and computational tools that are grounded on sound ontological foundations. In particular, we should advance in these disciplines a well-tested body of knowledge in terms of Ontology Patterns, Ontology Pattern Languages and Ontological Anti-Patterns . This chapter focuses on the latter.

An anti-pattern is a recurrent error-prone modeling decision [15]. In this paper, we are interested in one specific sort of anti-patterns, namely, model structures that, albeit producing syntactically valid conceptual models, are prone to result in unintended domain representations. In other words, we are interested in configurations that when used in a model will typically cause the set of valid (possible) instances of that model to differ from the set of instances representing intended state of affairs in that domain [11]. We name these configurations *Anti-Patterns for Ontology-Driven Conceptual Modeling*, or simply, *Ontological Anti-Patterns*.

In this chapter, we focus on the study of Ontological Anti-Patterns in a particular conceptual modeling language named OntoUML [11]. OntoUML is an example of a conceptual modeling language whose meta-model has been designed to comply with the ontological distinctions and axiomatization of a theoretically

well-grounded foundational ontology named UFO (Unified Foundational Ontology) [11]. UFO is an axiomatic formal theory based on theories from Formal Ontology in Philosophy, Philosophical Logics, Cognitive Psychology and Linguistics. OntoUML has been successfully employed in a number of industrial projects in several different domains, such as petroleum and gas, digital journalism, complex digital media management, off-shore software engineering, telecommunications, retail product recommendation, and government [12].

This chapter can be seen as complementary to our earlier work [23]. However, while previously we have focused on anti-patterns that are connected to the modeling of relations, here we focus on anti-patterns that emerge in connection to the modeling of *role types*, i.e., anti-rigid and relationally dependent types [11]. Stereotypical examples of role types include husband, student, employee, president, wife, etc. For instance, student is an anti-rigid type, i.e., a type that classifies its instances contingently. In other words, no student is necessarily a student (in a modal sense) and every instance of student can cease to be a student without ceasing to exist. Moreover, in order to be student, someone must be enrolled in an educational institution, i.e., being a student is a type of relationally dependent property.

The contributions of this chapter are three-fold. Firstly, we contribute to the identification of three new Ontological Anti-Patterns for Conceptual Modeling, in general, and for OntoUML, in particular. We do that by carrying out an empirical qualitative approach over a model benchmark of 54 OntoUML models. Secondly, after precisely characterizing these anti-patterns, we propose a set of refactoring plans that can be applied to the models in order to eliminate the possible unintended consequences induced by the presence of each of these anti-patterns. Finally, we present an extension for Menthon Editor, an OntoUML model-based editor with a number of features for: (i) automatically detecting anti-patterns in user models; (ii) supporting the user in exploring the consequences of the presence of an anti-pattern in the model and, hence, deciding whether that anti-pattern indeed characterizes a modeling error, either because it allows unintended model instances or because it forbids intended ones; (iii) automatically executing refactoring plans to exclude these unintended model instances, which can take the form of Object Constraint Language (OCL) [18] constraints or direct interventions in the model.

The remainder of this chapter is organized as follows: in Section 8.2, we briefly elaborate on the modeling language OntoUML and some of its underlying ontological categories; Section 8.3 discusses the notion of Ontological Anti-Patterns as taken in this work with some brief discussion of related work; Section 8.4 discusses the anti-pattern elicitation method employed here and characterizes the model benchmark used in this research; Section 8.5 presents the elicited Ontological Anti-Patterns with their unintended consequences, as well as possible solutions for their rectification in terms of model refactoring plans; Section 8.6 discusses empirical data about the occurrence of these anti-patterns in our model repository and presents the results of an additional industrial empirical study aimed at investigating the accuracy of our anti-pattern catalog as well as the efficacy of the proposed refactoring plans; Section 8.7 elaborates on the extensions implemented in the OntoUML editor taking into account these anti-patterns; Finally, Section

8.8 presents some final considerations.

8.2. Ontological Foundations

OntoUML is a language for Ontology-driven Conceptual Modeling, whose metamodel was designed to comply with the Unified Foundational Ontology (UFO) [11]. OntoUML has constructs that represent the ontological distinctions put forth by UFO as well as constraints on how these constructs can be combined. These OntoUML metamodel constraints are derived from the UFO axiomatization of the respective ontological distinctions. In the sequel, we briefly introduce some few OntoUML constructs that are germane for the purposes of this article. For a complete presentation and formal characterization of the language, the reader is referred to [11].

One of UFO's fundamental notions is that of rigidity. Rigidity is a property of types (i.e., a meta-property) that specifies whether its instantiation is essential or accidental for its instances. Rigid types are the ones that aggregate essential properties to all its instances, requiring them to instantiate it while existing (e.g. Car, Forest, Person, Band). Anti-rigid types, in contrast, aggregate accidental properties for its instances. If an individual instantiates an anti-rigid type in a given situation, there is at least one other alternative situation where it does not do so (e.g. Adult, Student, Customer, Husband).

In this paper, for the sake of conciseness in our presentation, we restrict ourselves to a fragment of OntoUML focused on sortal types. A sortal type is a type that carries a uniform principle of identity for its instances. Rigid sortal types, as the name suggests, are rigid types whose instances follow a unique principle of identity (e.g person, man, dog). A particular type of rigid sortal is the substance sortal type, which supplies a principle of identity for its instances (e.g. person, organization, car). In UFO and, hence, in OntoUML, substances sortals are either kinds, collectives or quantities. Since the examples used in this chapter are centered around functional entities of everyday experiences (e.g., people, organizations, computer systems), focusing on the notion of kind will suffice and will not cause a loss of generality. Still within the category of rigid sortals, we have rigid types that specialize substance sortals. These are termed *subkinds* (e.g. man, woman, Brazilian). Among the category of anti-rigid sortals, we have phases (e.g. adult, puppy, living) and roles (e.g. employee, student). In summary, in OntoUML, a sortal type is either a substance sortal (e.g., a kind) or a subtype of unique substance sortal (e.g. a subkind, role or phase). For more information, please see [11].

For this chapter, the notion of «role» is central. A Role is a type that is not only anti-rigid but also relationally dependent, i.e., a type whose instantiation depends on individuals bearing relational properties of certain sorts. In other words, in order to instantiate (play) a certain role, an individual must participate in certain material relations. Examples of roles and their respective relational dependencies include: a student and its enrollment in a school, an employee and its employment contract with a company, and a father with his legal children.

UFO (and, hence, OntoUML) distinguishes between two broad categories of relations, namely formal and material relations. In order for a set of individuals

to participate in a material relationship (i.e., a relation instance), they must be connected (mediated) by yet another entity termed a relator. A relator is said to be the truthmaker of that material relationship [10]. Take for instance the relation studies-at that can hold between a student and an university. For it to be true that Camila studies at University of Trento, there must be a particular enrollment connecting them (the relator). Other examples of material relations and their respective relator types are married-to and marriage, works-at and employment, and being-the-president-of and presidential mandate.

In OntoUML, we represent material relations using the «material» stereotype, and relator types are marked with the «relator» stereotype. Relations stereotyped as «mediation» are used to connect the related types (typically «role» classes) to the corresponding relator type (stereotyped as «relator»). Finally, the «derivation» relation is a formal relation connecting a material relation to the relator type whose instances are the truthmakers of the instances of that material relation (e.g., it connects the married-with relation to the relator type Marriage such that, for instance, married-with(Tiago,Camila) holds iff there is an instance of Marriage mediating the two).

Relators are examples of existentially dependent entities in UFO and OntoUML. In fact, they are examples of multiply existentially dependent entities, i.e., entities that depend on a multitude of individuals [11]. They belong, however, to a broader class of existentially dependent entities termed moments (also termed tropes, abstract particulars, objectified properties or particularized properties) [11]. So, relators are multiply dependent moments. In UFO; a moments that is existentially dependent on single entity is either a mode (e.g., a symptom, a belief, a disposition) or a quality (e.g., the color of an apple, the weight of a person) [11].

8.3. What is an Anti-pattern in Ontology-driven Conceptual Modeling?

Inspired by the Gang of Four's seminal work on design patterns [9], Andrew Koenig coined the term “anti-pattern” [15]. His original definition states that an anti-pattern is just like a pattern, in the sense that it is a recurrent solution for a given problem, but it is one that entails more bad consequences than good ones. To an anti-pattern, a proper pattern is to be associated as a better solution to the problem at hand. “The Blob” is an example of an anti-pattern for object-oriented software design [5]. The authors of that anti-pattern describe it as a procedural style design in an object-oriented environment, which cause one or few classes to aggregate most of the functionalities of the system, while the remainder classes just carry basic data or perform simple tasks.

In the late 90's, Fowler introduced the notion of *code smells* (or bad smells) [8]. The concept of code smell stands for a distinct code structure that requires careful attention because it is likely to produce maintainability and comprehensibility issues on the software being developed. The name *smell* conveys the idea of something “fishy” regarding the code. In their proposal, code smells are guides for code refactoring. The most basic example of a code smell presented by Fowler

is the “Duplicate Code”. Its definition is straightforward, a given code structure repeated in different places throughout a larger body of code.

In our work, we bring the ideas of anti-patterns and code smells to ontology-driven conceptual modeling, in general, and to OntoUML, in particular. Instead of software development, we deal with the problem of accurately capturing and formalizing a given conceptualization into a domain conceptual model. Instead of dealing with code structures, we deal with model structures. The idea is that “fishy model structures”, in this in this context, lead to domain misrepresentations from an ontological point of view.

We define an ontological anti-pattern as a modeling pattern that, despite producing syntactically valid conceptual models, it is prone to be the source of domain-related ontological misrepresentations. An anti-pattern must have a defined structure and refactoring options (or rectification plans) associated to it. Note that our definition combines the ideas behind anti-patterns and code smells. On one hand, we identify structures that capture recurrent modeling decisions that are prone to decrease model quality. Moreover, we combine these structures with appropriate solutions. In this spirit, our anti-pattern definition resembles the original one of Koenig. On the other hand, because our anti-patterns point to decisions that are not always incorrect and mean to serve as a guide for modelers to validate (and possibly refactor) their models, they also resemble code smells. We emphasize that our notion of anti-patterns is not as synonym for a modeling error or a bad modeling practice. One should think of it as model fragments that are worth “putting under the microscope”. On one hand, unlike Koenig’s definition of anti-pattern, the modeling solution at hand is not necessarily a bad one. On the other hand, unlike code smells, the decisions do not imply maintainability or architectural issues.

We should also differentiate an anti-pattern from a recurrent modeling expression that when created in a particular language always represents an error. Take, for instance, the situation in which an anti-rigid type is represented as a supertype of a rigid type, i.e., a type that classify its instances necessarily (e.g., the natural kind Person). This situation always leads to a logical contradiction [11]. As such, this is not an example of an anti-pattern, it is a logical (and ontological) mistake. In order to avoid this mistake, a language should include a syntactical constraint in its metamodel to always deem such a situation as syntactically invalid (i.e., as a syntactical error). This is exactly what is done in OntoUML [11], in which the incorporation of ontological constraints in its meta-model prevents the representation of ontologically non-admissible states of affairs [11], i.e., representations that would contradict UFO’s *domain-independent* axiomatization. However, as discussed in [11] there is no way a domain-independent language can exclude in a *prioristic* manner models that admit as instances representations that contradict *domain-specific axiomatizations*, i.e., instances that represent unintended state of affairs according to domain-specific conceptualizations [11].

To illustrate this point, suppose a conceptual model representing a transplant. In this case, we have domain concepts such as Person, Transplant Surgeon, Transplant, Transplanted Organ, Organ Donor, Organ Donee, etc. Suppose that this model has as a possible instance one that represents a state of affairs in which the Donor, the Donee and the Transplant Surgeon are one and the same Per-

son. Now, suppose that this instance represents an unintended state of affairs according to our assumed conceptualization of the domain of transplants. Notice that the state of affairs represented by this instance is only considered inadmissible (unintended) due to domain-specific knowledge of social and natural laws. As discussed in [11], the presence of this unintended instance is typically caused in OntoUML models by the introduction of a particular model fragment. Such a model fragment would be an example of what we term here an Ontological Anti-Pattern.

8.3.1. Related Works: Anti-Patterns in the Semantic Web

Since Koenig's original proposal [15], the concept of anti-pattern has been applied in a variety of fields other than software design. We are unaware of other works in the literature that systematically study anti-patterns for ontology-driven conceptual modeling. For this reason, we compare our take on anti-patterns to related in the context of the semantic web. As it shall become clear in the sequel, when referring to anti-patterns or similar notions, these works assume an interpretation of the term that differs significantly from the one employed here.

In [20], Roussey et al. focus on OWL anti-patterns. For them, an OWL anti-pattern is a pattern that is commonly used by domain experts in their OWL implementations and that normally result in inconsistencies. They argue that anti-patterns come from a misuse and misunderstanding of description logics expressions by ontology developers. Their anti-patterns are classified in three exclusive categories: (i) logical, which represents errors that reasoners detect; (ii) cognitive, which classifies possible modelling errors that are not detected by reasoners; and (iii) style (originally named “Guidelines”), which describes logically correct expressions made unnecessarily complex, e.g. using the complement operator instead of disjointness. Our approach differs from theirs in the nature of the problems we investigate. First, we are not concerned with inconsistency issues here because the most typical logical mistakes that are of an ontological nature are prevented by OntoUML's metamodel by design. Moreover, we are also not concerned with anti-patterns as conventional guidelines in the same spirit because we want to improve model precision and accuracy, not readability or maintainability. Lastly, cognitive anti-patterns include problems like expression redundancy, which is also out of the scope of our anti-pattern validation approach.

Poveda-Villalón et al. build their validation approach [19] around the concept of common ontology pitfalls. A pitfall stands for an anomaly or worst practice in ontology design identified through empirical analysis. As Roussey et al., they also see these recurrent problems as misunderstanding and misusing the description logics constructs. The authors classify the common pitfalls identified in RDF and OWL ontologies, from two different perspectives. First, regarding the type of problems they can cause, the pitfalls are classified into structural, functional and usability pitfalls; second, regarding the quality criteria they affect, they are classified into the categories of consistency, completeness and conciseness pitfalls. The use of pitfalls for ontology validation is supported by a tool named OOPS! Ontology Pitfall Scanner [19]. In our work, we require anti-pattern definitions to prescribe finite and identifiable model structures. In contrast, many pitfalls

proposed in OOPS! do not have this property. An example is the pitfall “Merging different concepts in the same class”, which rely on an analysis of class names, to identify the use of the operators “and” or “or” on the class name.

Baumeister and Seipel investigate recurrent anomalies in OWL ontologies enriched with SWRL rules [4]. They distinguish between four types of anomalies, namely circularity, inconsistency, redundancy, and deficiency. Circularity refers to circular definitions in the ontology, involving either classes, properties and rules, that hinder the performance of reasoners. The inconsistency category describes anomalies that produce actual logical contradictions in the ontology. A very common example, according to the authors, is the definition of a class as subclass of two disjoint classes. Anomalies related to redundancy point out knowledge that can be excluded from the ontology without changing its semantics. Lastly, deficiency anomalies refer to problem of completeness, understandability and maintainability of the ontology. An example of deficiency is the “lazy class”, which stands for classes that have little use in practical applications of the ontology.

In summary, our approach differs from all the aforementioned efforts on a very key aspect. We do not intend that our anti-patterns support modelers in building the models correctly, but in building the (ontologically) correct model for the domain. In other words, our focus is on model validation instead of model verification [11]. Because of that, we do not focus on modeling patterns that are always wrong (e.g. patterns that lead to logical inconsistency). Instead, we focus on anti-patterns that represent structures that frequently (as empirically demonstrated) cause dissociations between the set of valid instances admitted by a model and the set of intended instances admitted by the underlying domain conceptualization [11].

8.4. Empirically Uncovering Ontological Anti-patterns

8.4.1. Method

Our approach to identify ontological anti-patterns is an empirical qualitative analysis. It starts with the selection of a particular model for analysis. Within the selected model, the second step is to select relevant fragments for analysis. Such fragments can consist of a whole diagram, a subset of a diagram or even a new “artificial” diagram produced for the sake of analysis (model inspection).

Step three is to inspect the selected portion of the model and identify possible problems. We conduct this activity using visual model simulation [12]. This simulation consists in converting OntoUML models into Alloy [14] specifications, generating possible model instances and contrasting these instances with the set of intended instances of the model. The set of intended instances correspond to those that represent intended state of affairs [11] according the creators of the models. Upon the identification of a mismatch, we register it as a potential problem.

After detecting a possible problem, we analyze the model in order to identify which structures (i.e., combination of language constructs) caused that problem. In the sequence, we interact with the modelers (when available) or inspect the

documentation accompanying the model to define whether the identified structure is indeed problematic. If it is, we propose a possible solution to rectify the model and register it as a (problem, solution) pair. With a recently modified model, we go back to step three. This iteration is repeated until no more problems can be identified in the fragment and then, another fragment is selected. The analysis stops whenever we inspect all relevant model fragments. After inspecting each model, we analyze the generated problem-solution pairs in order to generalize them into pairs of anti-patterns and refactoring plans.

8.4.2. Model Repository

Our empirical analysis for uncovering anti-patterns was performed using a repository of 54 models¹. Tables 8.1 and 8.2 provide an summary on the models our repository according to the following dimensions:

- **Name:** the name provided by the model's authors. If none was provided, we baptized the model with a chosen intuitive name.
- **Context:** the scenario in which the model was developed. The model is placed in one of the following categories: Government Project, for models developed by or in cooperation with governmental entities; Industrial Project, for models produced by or in cooperation with private companies; Graduate course assignment (assignment) for models produced by graduate students as a final assignments of an “Ontology Engineering” 60-hour course offered by the Graduate School on Informatics of the Federal University of Espírito Santo; PhD Thesis (PhD), MSc Dissertation (MSc) and BSc Monograph (Bsc), for models produced as outcome of academic research in each of these respective levels; and Other, for the models that do not fit the aforementioned categories.
- **Purpose:** describes the motivation for building the model at hand. The category termed interoperability is intended for those models created as reference models for semantic interoperability between agents and/or systems; ontological analysis (ontol. analysis) classifies models developed to evaluate conceptual modeling languages or other domain formalizations; reference model (reference model) classify those models that are proposed as reference ontologies for a given community; knowledge-based application (kb application) stands for models created to support the development of this kind of applications; lastly, unspecified categorizes models whose authors did not provide further characterizing information on this topic.
- **Expertise (Exp.):** describes the authors' familiarity to the OntoUML language during the development of the ontology. We classify a modeler as a beginner (beg) if he/she had at most a year of experience using the language, and as advanced (adv) otherwise.
- **Number of Participants (Part.):** provides an estimated number of people involved in the production of the ontology. We consider as participants

¹The conceptual models in this table are publicly available at: <http://www.menthor.net/model-repository.html>. The few exceptions of missing models are due to non-disclosure agreements that prohibited their publication.

those actively involved in either scope definition, knowledge acquisition, design or evaluation. For published models, we estimate the number of participants using the authors of the publication, whilst for those models developed in the context of theses, we consider the student and all official supervisors.

Table 8.1 lists the 32 models developed as graduate course assignments, whilst Table 8.2 present the remainder 22. From these, 11 models were developed during academic research without industry collaboration. An example is The Configuration Management Task Ontology [6], a product of a Masters dissertation. Furthermore, seven models had total or partial participation of private companies and/or governmental organizations. The most significant one is the MGIC Ontology [3], developed in the context of a research project with a regulatory agency responsible for controlling ground transportation services in Brazil. Finally, the development contexts of 4 models were not available.

Concerning the purpose for which the models have been created, the repository contains 10 models (16%) that are intended to serve as a reference domain or core ontologies (e.g. UFO-S [17] for the domain of services). Another 10 models (16%) have been developed in order to perform ontological analysis on existing formalizations, databases or modeling languages. An example is the refactoring of the Conceptual Schema of Human Genome presented in [16]. The repository also contains 8 models (13%) designed for supporting the development of knowledge-based applications, 6 (10%) whose main intention was to support semantic interoperability between systems and/or organizations, and only two (3%) for the purpose of enterprise modeling. For the remainder 26 models (42%), there is no information w.r.t. this classification dimension.

Regarding the modeler's overall expertise in OntoUML, 22 models (41%) have been developed by beginners (18 of these models are also graduate course assignments) and 32 (59%) developed by experienced modelers. Finally, we look into the total number of modelers involved in the model construction. A single person participated in the development most of the models (35 models, roughly 65%). However, 15 models were the product of collaboration efforts between 2-4 people and 4 models involved 7-10 conceptual modelers.

8.5. Ontological Antipatterns

In this section, we extend the anti-pattern library presented in [13, 22, 23] with three anti-patterns centered around OntoUML's «role» construct. In order to facilitate learning and usage of these anti-patterns, we describe them following a consistent format. First, we discuss each anti-pattern in natural language. Next, for each anti-pattern, we present a table that summarizes the anti-patterns essential characteristics. Lastly, we exemplify that anti-pattern with one its occurrences in the models presented in the model repository considered in this study. For each anti-pattern, we then describe the following information:

- **Name and Acronym:** unique identification in the catalogue;
- **Description:** a general description of the anti-pattern;

Table 8.1. Overview of models in the repository developed as assignments.

Model	Context	Main Purpose	Exp.	Part.
Gi2MO Refactored	assignment	ontol. analysis	adv	1
Internal Affairs Ontology	assignment	ontol. analysis	adv	3
Library Model	assignment	unspecified	beg	1
Public Tenders Model	assignment	unspecified	adv	1
Social Contract Model	assignment	unspecified	beg	1
Clergy Model	assignment	unspecified	beg	1
FIFA Football Model	assignment	unspecified	adv	1
IDAF Model	assignment	unspecified	adv	1
University Model	assignment	unspecified	beg	1
GRU MPS.BR Model	assignment	unspecified	beg	1
Experiment Model	assignment	unspecified	beg	1
Parking Lot System	assignment	unspecified	adv	1
Quality Assurance Ontology	assignment	unspecified	adv	1
Music Ontology Refactored	assignment	ontol. analysis	adv	1
Internship Model	assignment	unspecified	adv	1
G.809 Model	assignment	unspecified	beg	1
Online Mentoring Model	assignment	unspecified	adv	1
Help Desk System Model	assignment	unspecified	beg	1
IT Infrastructure Model	assignment	unspecified	beg	1
Photography Model	assignment	unspecified	beg	1
FIRA Ontology Refactored	assignment	onto analysis	adv	1
Banking Model	assignment	unspecified	adv	1
Chartered Service Model	assignment	unspecified	adv	1
Health Organization Model	assignment	unspecified	beg	1
Bank Model 2	assignment	unspecified	adv	1
PROV Ontology Refactored	assignment	ontol. analysis	beg	1
WSMO Refactored	assignment	ontol. analysis	beg	1
Recommendation Model	assignment	kb application	beg	1
Project Management Model	assignment	unspecified	beg	1
Construction Model	assignment	unspecified	beg	1
Stock Model	assignment	unspecified	beg	1
Real State Model	assignment	unspecified	beg	1

Table 8.2. Overview of models in the repository developed in various contexts.

Model	Context	Main Purpose	Exp.	Part.
OpenFlow Ontology	BSc	kb application	beg	2
CMTD	MSc	interoperability	adv	2
OntoEmerge	MSc	kb application	adv	8
PAS 77:2006 Ontology	MSc	ontol. analysis	adv	4
Cloud Vulnerability Ontology	MSc	reference model	adv	4
OntoUML Org Ontology	MSc	ontol. analysis	adv	2
ECG Ontology	MSc	interoperability	adv	3
Requirements Ontology	MSc	kb application	adv	2
Open proVenance Ontology	PhD	reference model	adv	3
CSHG Refactored	PhD	ontol. analysis	adv	3
UFO-S	PhD	reference model	adv	7
MGIC Ontology	government	enterprise model	adv	10
MPOG Ontology Draft	government	reference model	beg	7
OntoBio	government	interoperability	adv	3
Normative Acts Ontology	government	reference	adv	3
G.805 Ontology	industry	ontol. analysis	adv	4
G.805 Ontology 2.0	industry	kb application	adv	3
G.800 Ontology	industry	kb application	adv	3
TM Forum Model	other	unspecified	beg	1
School Transport Model	other	application	beg	1
ERP System Model	other	interoperability	adv	1
Inventory System	other	interoperability	adv	1

- **Structure:** structural definition of an anti-pattern, consisting of (i) *pattern roles*, to describe the elements that participate in the anti-pattern, their possible stereotypes and cardinalities; and (ii) a *generic example*, i.e., a graphical exemplification of the anti-pattern's generic structure;
- **Refactoring Plan:** recurrent alternative solutions defined as a sequence of actions that can be used to rectify the model containing the problem created by the occurrence of that anti-pattern. Includes: (i) *constraint definition*, to indicate the specification of an OCL constraint; (ii) *element modification*, to indicate a change in a model element (e.g. meta-property change); and (iii) *element creation*, to indicate the introduction new elements to the model.

Note that mediation (and characterization) relations always have a minimum cardinality of at least 1 and an immutability meta-property (i.e., *readOnly = true*) in their target association end, i.e., on the association end of this relation connected to the mediated (characterized) type. For the sake of brevity, we refrain from repeating these constraints in each depiction of these relations when presenting anti-pattern structures.

8.5.1. Relator Mediating Rigid Types

The first anti-pattern we introduce here is called *Relator Mediating Rigid Types* (*RelRig*). Its occurrence is characterized by a «relator» class *R* connected to one or more rigid sortal classes $T_1..T_n$ (e.g. «kind», «subkind») through «mediation» relations. *RelRig*'s main characteristics are presented in Table 8.3.

A type T_i is deemed relationally dependent if it is connected to a «relator» *R* through a «mediation» *m*. In other words, for an individual x to be an instance of type T_i there must exist an instance of *R* that mediates x [11]. For instance, *Student* is typically a relationally dependent type whose instances are mediated by instances of *Enrollments*, i.e., in order for someone to be a *Student*, she must be mediated by a particular *Enrollment*. Notice that in this example, *Student* is a «role». The issue at stake with this anti-pattern is that rigid sortal types aggregate essential properties of individuals and thus are usually relationally independent types [11]. Mediation relations, in contrast, are commonly used to define roles, i.e., anti-rigid types that aggregate accidental relational properties.

In our empirical investigation, we confronted ontologists regarding the models in which they judged that this anti-pattern really introduced an error (i.e., an unintended representation). Our goal was to find out which was the original modeling intention that caused (by mistake) the introduction of this anti-pattern in the model.

- (i) the mediated type T_i at hand should actually be modeled as a role. For instance, suppose one represents a type Public Enterprise as a «subkind» of Enterprise that is connected to a «relator» type Public Offering. In this case, if an Enterprise can exist before going public, it means that being a Public Enterprise is a contingent (and relationally dependent) subtype of Enterprise;

Table 8.3. Summary of the *RelRig* anti-pattern

Name (Acronym)	Description	
Relator Mediating Rigid Types (RelRig)	A relator connected to one or more rigid types through mediations. This is a potential problem because relational dependencies are commonly defined for anti-rigid types.	
Pattern Roles		
Mult.	Name	Allowed Metaclasses
1	R	«relator»
$1..^*$	m_i	«mediation»
$1..^*$	T_i	«kind», «quantity», «collective», «subkind»
Generic Example		
Refactoring Plans		
<p>1. Role: Change T'_i's meta-class to role.</p>		
<p>2. Role subtype: Create a role that specializes T_i and reconnect the respective m_i to it.</p>		
<p>3. Reversed dependency: a) change T_i to mode and M_i to characterization or b) change T_i to relator and invert m_i.</p> <p>(a) (b) </p>		
<p>4. Bidirectional dependency: transform R into a sortal and change m_i into an <i>essential</i> and <i>inseparable</i> parthood relation.</p>		

- (ii) the «mediation» relation was not supposed to be declared with a mandatory cardinality constraint from the perspective of T_i (i.e., in the association end opposite to T_i in the «mediation» relation). Now, if this cardinality constraint is an optional one, this means that there is a subtype ST_i of T_i that has been omitted in the model. Notice that ST_i is defined as the type instantiated by the instances of T_i when mediated by instances of the «relator» R at hand. As such, ST_i is by definition a «role» since: it is anti-rigid - the instances of ST_i can exist (as instances of T_i) without being instances of ST_i ; instances of T_i become instances of ST_i when mediated by instances of R . As an example of this situation suppose that someone represents a constraints stating that instances of *Person* are necessarily connected to a *Citizenship* (a «relator»). After inspection, the modeler can realize that actually there are people that do not have a *Citizenship* (e.g., people that were born but not yet formally registered) and that there is a subtype of *Person* (namely *Citizens*) that have a *Citizenship* connecting them to a *Country*;
- (iii) there is indeed an existential dependency that should be represented in the model but in the reverse direction, i.e., it is the type T_i that has instances that are existentially dependent on corresponding relators, not the other way around. This means that this otherwise mediated type is not a sortal type but a type whose instances are moments. This case has still two variants:
 - (a) the case in which T_i is a type whose instances are modes, in which case the «mediation» relation between T_i and R should be replaced by a «characterization» relation. As an example, take the case in which we have a *Bundle of Rights* that one has in the scope of a particular *Employment*;
 - (b) the case in which T_i is a type whose instances are relators, in which case the mediation relation between T_i and R is indeed a mediation relation but it is T_i that connects instances of R to instances of another type and not the other way around. As an example, suppose we have a *Chairmanship* as a *Parliament Commission Chairman*, which existentially depends on a *Mandate* «relator» (as well as on a specific *Commission*). As this example shows, finding out that *Chairmanship* is actually a relator triggers a process of instantiating the entire Relator Pattern [12], which in turn leads to the possible discovery of the role *Parliament Commission Chairman* (a «role» played by instances of the «role» *Congressman*) as well as the type *Commission*;
- (iv) finally, we have a case in which there are indeed mutual existential dependence relationships between instances of T_i and instances of R . However, as discussed in depth in [11], sortals are not existentially dependent on entities from which they are mereologically disjoint. In other words, if a sortal x is existentially dependent on sortal y then either x must necessarily be a part of y (which we term an inseparable part) or y must necessarily be a part of y (which we term an essential part) (see [11] for details). As an example of this case, suppose that one wants to model that a *Social Contract* R creates

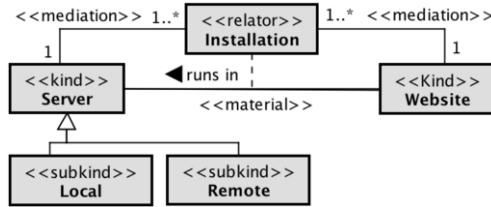


Figure 8.1. *RelRig* example extracted from the PAS-77 Ontology [7].

a *Joint Venture* between a number of organizations $T_1..T_n$ that only exist for the scope of that transient *Joint Venture*, i.e., once the social contract ceases to exist then the organizations themselves cease to exist. In this case, the relation between an *Organization* instance of T_i and an instance r of R is not one of mediation but one of inseparable and essential parthood. In other words, r is defined as a mereological sum of instances of T_i and each instance of T_i only exist as part of that *Joint Venture* r .

Figure 8.1 presents a model fragment adapted from the PAS-77 Ontology [7] that contains a *RelRig* occurrence. This simplified fragment states that a *Website* has to be installed in one or more *Servers*. It also states that a *Server* has at least one *Website* running on it. Finally, the authors distinguish *Servers* with regard to where they are located, i.e., within a company or in a remote location.

As it is, the model does not account for recently acquired or formatted computers, which would not have any application running on them. If this is a desired possibility, the authors have improperly characterized *Server* as a «kind». Instead, it is a «role» a *Computer System* play when hosting an application or a website. Regarding the representation of *Website*, the model forbids the existence of instances of website that are not installed in any server. If we consider that a *Website* exists since its development, when it is not running anywhere, then we may want to include a new «role» in the model, namely *Active Website*.

8.5.2. Free-Floating Role Specialization

The *Free-Floating Role Specialization (FreeRole)* anti-pattern focuses on uncovering (possibly) missing conditions for certain roles to be instantiated. Its generic structure consists of: a «role» Ro connected to relators $R_i..R_n$ through mediations $m_i..m_n$; the «role» Ro is specialized by one or more roles $SRo_i..SRo_n$; the specializing $SRo_i..SRo_n$ are not directly connected to any additional «mediation» relations. The essential features of the *FreeRole* anti-pattern are summarized in Table 8.4.

Our empirical investigation suggests that *FreeRole* occurrences are often modelling errors, which can be corrected by applying one of the following role specialization refactoring plans:

- (i) *Derived sub-role*: this should be applied when SRo_i is instantiated according to a derivation rule. For example, a *Person* plays the role of *Student* when

Table 8.4. Summary of the *FreeRole* anti-pattern.

Name (Acronym)	Description	
Free Role Specialization (FreeRole)	A role connected to a relator through a mediation, is specialized by one or more roles, which, in turn, are not connected to additional mediation relations. This pattern suggests that a properly characterized instantiation criteria for these specializing roles might be missing.	
Pattern Roles		
Mult.	Name	Allowed Metaclasses
1	Ro	«role»
1..*	m_i	«mediation»
1..*	R_i	«relator»
1..*	SRo_i	«role»
Generic Example		
<pre> graph TD Ro["<<role>> Ro"] --- m_i["<<mediation>> m-i"] Ro --- SRo_i["<<role>> SRo-i"] Ro --- SRo_j["<<role>> SRo-j"] m_i --- R_i["<<relator>> R-i"] SRo_i --- SRo_i SRo_j --- SRo_j </pre>		
Refactoring Plans		
<p>1. Derived sub-role: add proper OCL constraint defining the derivation rule at hand.</p> <p>2. Role-of-a-role: create a new relator R and connect it to SRo_i using a new mediation relation.</p>		
<pre> graph TD Ro["<<role>> Ro"] --- m_i["<<mediation>> m-i"] Ro --- SRo_i["<<role>> SRo-i"] SRo_i --- R["<<relator>> R"] R --- m["<<mediation>> m"] m --- SRo_i </pre>		
<p>3. Intentional sub-role: create a new relator SR_i by specializing R_i and connect it to SRo_i using a new mediation.</p>		
<pre> graph TD Ro["<<role>> Ro"] --- m_i["<<mediation>> m-i"] Ro --- SRo_i["<<role>> SRo-i"] SRo_i --- SR_i["<<relator>> SR-i"] SR_i --- m["<<mediation>> m"] m --- SRo_i </pre>		

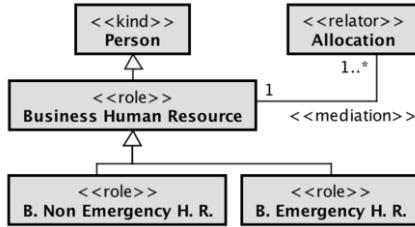


Figure 8.2. *FreeRole* example extracted from the OntoEmerge Ontology.

she enrolls at an *Educational Institution*; *Students* are classified as *Freshmen*, if enrolled for less than a year, and as *Seniors* otherwise;

- (ii) *Role-of-a-role*: this should be applied when SRo_i is instantiated due to an additional relational dependency. To exemplify, we use the *Student* concept once more. A *Student* becomes an *Intern* when she starts an *Internship* program (a new relator) within a *Company*;
- (iii) *Intentional sub-role*: this is suggested when SRo_i is instantiated due to a particular subtype of the generic dependence relation characterizing role Ro . As an example, consider the concepts of *Bachelor Student* and *Graduate Student*. These are both types of *Student*, and thus, both are defined by *Enrollments*. However, each of these student roles requires particular subtypes of *Enrollments* with particular characteristics. For instance, unlike a *Bachelor Enrollment*, a *Graduate Enrollment* requires the assignment of a *Thesis Supervisor*.

The small model fragment depicted in the right side of Figure 8.2 corresponds to a *FreeRole* occurrence encountered in OntoEmerge [2], an ontology for the emergency domain. In this current presentation, the model accepts that for a given allocation, a person can play an emergency-related role at a given time and a non-emergency one at another, without any apparent property being changed in the allocation itself. The intentional sub-role pattern can then be used to provide a more suitable representation for this domain. In this case, two subtypes of the relator would be created, one for each type of business resource role.

8.5.3. Multiple Relational Dependency

Roles are relationally dependent types. This means that an individual plays (or instantiates) a role whenever a particular type of material relation is established with another individual. OntoUML, however, does not impose any constraint with respect to the number of material relations that can be used to characterize a role, each possibly representing a different source of relational dependence. The *Multiple Relational Dependency* (*MultDep*) anti-pattern focuses the attention of the modeler on types whose definition includes multiple material relations characterizing a role. Structurally, its identification consists of a sortal class directly

connected to two or more «mediation» relations. Table 8.5 provides an overview of the *MultDep* anti-pattern.

A *MultDep* occurrence requires attention because, through our empirical analysis, we identified that it might indicate:

- (i) The model might be over-constrained, in the sense that one or more «mediation» relations represented as mandatory relations are in fact optional relations. This could in itself can potentially hide other implicit role types;
- (ii) In addition to (i), there is a particular order in which two or more optional «mediation» relations can be established;
- (iii) The model is lacking a relationship between two or more of the «relator» types that characterize the multiple sources of relational dependence.

The first step to analyze a *MultDep* occurrence is to verify whether the ascribed mediation relations are indeed mandatory. If that is not the case for a given mediation relation, the appropriate refactoring solution is the transformation of that mediation relation into an optional one. This is followed by the inclusion of a new role in the model, which would “carry” the mediation relation for itself. In the particular case in which a modeler concludes that two or more mediation relations are optional, an evaluation of whether they are established in a particular sequence is in order. To exemplify, consider two roles that can be played by a person: the student role, when related a school (via an enrollment relator), and the employee role, when related to a company (via an employment relator). A priori, one cannot make any assertion w.r.t the order in which one becomes a student and an employee. Conversely, if we consider the roles of student and intern (and their implied dependencies) there is clearly a restriction regarding the order in which these two roles should be played, namely, that a person could only become an intern if she is already a student. This means that the internship (a relator) connecting an intern to a company depends on the existence of an enrollment (another relator) connecting that intern as a student to a school.

If the class that characterizes the *MultDep* occurrence has more than two mediations connected directly to it, a modeler might refactor it by simultaneously defining ordered and unordered optional mediation relations. Nonetheless, when creating unordered dependencies, all new roles should specialize the same supertype. If an order is required, the modeler should create the new roles as subtypes of one another, i.e., they should obey a pre-defined hierarchical order.

Regardless if a *MultDep* occurrence identifies optional mediation relations or not, the next step is to analyze if there are relational dependence sources that depend on other relational dependence sources. In other words, if there are existential dependence relations between the relators that embody these different relational dependence sources. To illustrate this point, consider the following scenario: *A person becomes an undergraduate student when she enrolls in a bachelor program at a university, e.g. Computer Science or Philosophy. A unique number identifies each enrollment. Victor, a very curious and dedicated young man, decides to pursue, simultaneously, a major in Philosophy and Computer Science. To do that, he would need to enroll two times at the university. After his enrollments, Victor wants to apply for Logics 101 as a Computer Science major and*

Table 8.5. Summary of the *MultDep* anti-pattern.

Name (Acronym)		Description
Multiple Relational Dependency (MultDep)		A sortal class directly connected to two distinct relators through mediation relations. This structure might indicate redundancy, over constraining or scope issues.
Pattern Roles		
Mult.	Name	Allowed Metaclasses
1	T	All sortal types
$2..^*$	m_i	«mediation»
$2..^*$	R_i	«relator»
Generic Example		
<pre> graph LR T[T] <--> R_i["<<relator>> R-i"] T <--> R_n["<<relator>> R-n"] T -- "m-i" --> R_i T -- "m-n" --> R_n </pre>		
Refactoring Plans		
<p>1. Unordered optional dependency: For each selected m_i, create a direct subtype of T and connect m_i to it.</p> <pre> graph TD T[T] -- "m-i" --> Ro_i["<<role>> Ro-i"] T -- "m-n" --> Ro_n["<<role>> Ro-n"] Ro_i -- "m-i" --> R_i["<<relator>> R-i"] Ro_n -- "m-n" --> R_n["<<relator>> R-n"] </pre>		
<p>2. Ordered optional dependency: For each selected m_i, create a new role following the chosen instantiation order.</p> <pre> graph TD T[T] -- "Mediation1" --> Ro_i["<<role>> Ro-i"] T -- "Mediation2" --> Ro_n["<<role>> Ro-n"] Ro_i -- "m-i" --> R_i["<<relator>> R-i"] Ro_n -- "m-n" --> R_n["<<relator>> R-n"] </pre>		
<p>3. Relator dependency: For each identified dependency, create a mediation connecting the two relators.</p> <pre> graph LR Type[Type] <--> R_i["<<relator>> R-i"] Type <--> R_n["<<relator>> R-n"] R_i -- "m-i" --> Type R_n -- "m-n" --> Type R_i -- "dependsOn" --> R_n </pre>		

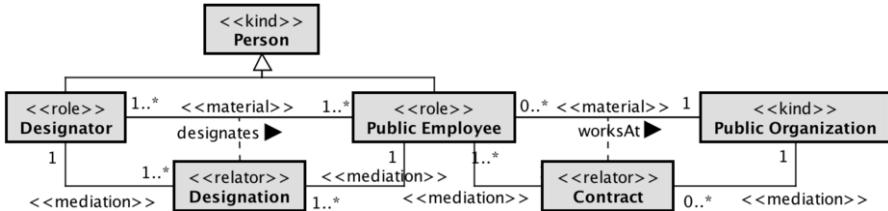


Figure 8.3. *MultDep* example extracted from the MGIC Ontology [3].

apply for Sociology 101 as a Philosophy major. To do that, each course application must not only identify Victor as the applying student, but also identify the particular enrollment he is using to apply.

The identification of the enrollment in the course application suggests that a course application is existentially dependent of the student, the course and the actual specific enrollment associated to a particular major. We propose the formalization of this dependency in an OntoUML model as a new «mediation» relation from course application to enrollment in a major.

The *MultDep* example we selected comes from the ontology developed in the MGIC project, whose goal was to develop a model for knowledge and information management for the Brazilian Ground Transportation Regulatory Agency (ANTT). The small model fragment depicted on Figure 8.3 states that a public employee has one or more designations (for a permanent position in the public administration) and exactly one valid contract. The sensitive issue here is that, according to the law governing these public positions in Brazil, a given contract is grounded on a particular designation (a part of the public employment process in Brazil). As presented in Figure 8.3, the model allows for an independent variation of contracts and designations, which should not be allowed. Refactoring plan 3 should be adopted to rectify this model, thus formalizing an existential dependency from contract to designation.

8.6. Anti-Pattern Evaluation

We define semantic anti-patterns as being error-prone recurrent modeling decisions. Therefore, in order to evaluate them, we analyze two dimensions. This first is frequency, which measures how recurrent the modeling decisions are. The second is usefulness, which indicates how error-prone the anti-patterns are. In this section, we report on the results of two studies we conducted to assess each of these aspects.

For the sake of clarity, we have only discussed anti-patterns that arise from role modeling in the case of sortals. However, these anti-patterns are generalizable to also take into account roleMixins. A roleMixin [11] is just like a role, but instead of being played by instances of the same kind (e.g. student-person), it is played by instances of different kinds. Examples of roleMixins include *Customer* and *Insured Item*, the former being played by both people and companies, whilst the latter being played by cars, houses, and paintings.

Table 8.6. Frequency evaluation results.

Anti-pattern	Occ.	M.Occ.	$\frac{M.Occ.}{AllMo}$
RelRig	282	37	69%
FreeRole	199	18	33%
MultDep	105	28	52%

We highlight that the following studies use these generalized versions of the anti-patterns presented in this chapter. Nonetheless, they are still useful assessments of how frequent and how problematic an anti-pattern is. For more details on the extended version of the anti-patterns, please refer to [21].

We also make emphasize that the version of the *RelRig* anti-pattern presented in this chapter is an evolution of the version used in the empirical studies reported here. The major improvement regards an alteration in the refactoring plan named "Bidirectional dependency". The previous solution suggested that modelers should keep the relation under analysis as a mediation and specify it as a bidirectional dependency. However, this would contradict UFO's definition of objects (substantial individuals), which prevents theses individuals from being existentially dependent of other individuals that are mereologically disjoint from them. To follow this constraint, the refactoring plan was modified to suggest the representation of bidirectional dependencies as part-whole relations.

8.6.1. Frequency Analysis

Our first study measured is anti-pattern frequency, i.e., how often modelers design structures that fit an anti-pattern definition. At this moment, we did not assess if the identified structures were indeed mistakes. To conduct the study, we implemented algorithms to automatically query the models for anti-pattern occurrences, which we later incorporated into the Menthon Editor.

To understand how frequent an anti-pattern occurrence is we organize the results in Table 8.6. The meaning of each column is the following: *Occ.* is the sum of all anti-pattern occurrences in all models of the analyzed repository; *M.Occ.* is the number of models with at least one given anti-pattern occurrence in the analyzed repository; $\frac{M.Occ.}{AllMo}$ represents the percentage rate between *M.Occ.* and all models (*AllMo*) in the repository (i.e., 54, in the case of this repository);

As Table 8.6 shows, the three anti-patterns identified here are indeed recurrent modeling decisions. *RelRig*, *MultDep* and *FreeRole* were identified 282, 105 and 119 times, respectively, throughout the 54 models. The fact that these modeling decisions can be found in a significant subset of the inspected models (as shown by $\frac{M.Occ.}{AllMo}$) also corroborates their recurrent nature. Among the three anti-patterns, *RelRig* is the most frequent one, found in 69% of the models, whilst *MultDep* was identified in a little more than half (52%) of the models, and *FreeRole* in roughly a third of them (33%).

Table 8.7. Usefulness evaluation results.

Anti-pattern	Occ.	Error	$\frac{\text{Error}}{\text{Occ.}}$	Refac.	$\frac{\text{Refac.}}{\text{Error}}$	Partial	Custom
RelRig	161	107	66.5%	105	98.1%	1	1
FreeRole	39	23	59.0%	19	82.6%	2	2
MultDep	41	23	56.1%	16	69.6%	6	1
Total	241	153	63.5%	140	91.5%	9	4

8.6.2. Usefulness Analysis

In this second study, we focus on anti-pattern usefulness. The goal is to measure two things: (i) the probability of an occurrence to characterize a domain misrepresentation, i.e., an actual modeling error; and (ii) how often we can predict the solution for an occurrence whenever a problem is identified, i.e., how often our rectification solutions to the problems at hand are adopted by the modelers.

It is not possible to conduct a study of this nature in a fully automatic way, since judging whether an anti-pattern occurrence is an error is a matter of domain knowledge. With that in mind, we chose to conduct the anti-pattern usefulness evaluation as a case study using the MGIC ontology [3]. The first reason we chose this ontology is its size: 3800 classes, 1918 associations, 3616 generalizations, 698 generalization sets, 71 data types, 865 attributes and 149 constraints. The ontology also contains occurrences of all anti-pattern types, having been developed by ten modelers throughout three years. Lastly, the ontology was a product of a governmental project and we had access to the modelers who produced it.

Eight modelers participated in the case study. We assigned sub-domains to each of them, taking into account their knowledge about the domain and its complexity (represented by the number of classes and relations used to formalize it). To guarantee that the modelers would have enough knowledge to analyze the anti-pattern occurrences, we mostly assigned parts of the ontology that they designed. We also encouraged modelers to interact with each other during the case study. Modelers conducted the anti-pattern detection and analysis exclusively using a modeling environment called Menthon Editor.

Together, the participants analyzed 241 occurrences of the anti-patterns presented in this chapter. We summarize the results in Table 8.7. The column *Occ.* stands for the number of analyzed occurrences of a given anti-pattern type, whilst the ones labeled as *Error* and $\frac{\text{Error}}{\text{Occ.}}$ refer to the total number and percentage of occurrences considered as modeling errors by the participants, respectively. The columns *Refac.*, *Partial* and *Custom* stand for the sum of occurrences the participants fixed using: (i) exclusively our proposed solutions; (ii) a variation of one of our proposed solutions; (iii) and exclusively custom solutions, respectively. $\frac{\text{Refac.}}{\text{Error}}$ presents the percentage of anti-pattern occurrences fixed exclusively using proposed refactoring plans.

By comparing the first two columns of Table 8.7, we have a positive indication that the proposed anti-patterns are indeed error prone structures. In all three cases, the participants considered more than half of the analyzed structures errors. Among the three, *RelRig* seems to be the most pernicious one, since two thirds

Table 8.8. Usefulness evaluation results.

Anti-pattern	Frequency	Problem Rate	Predictability
RelRig	High	High	Very High
FreeRole	Low	Medium	Very High
MultDep	Medium	Medium	High

of its occurrences were considered problematic.

This case study also provides positive evidence that the refactoring plans are actually viable recurrent solutions. *RelRig*'s refactoring plans were used in almost all scenarios in which a problem was identified, roughly 98% of the cases. *FreeRole*'s predefined solutions were a little less accurate, but still helped participants in 82.6% of the time. The least accurate refactoring plans belong to *MultDep*, even though the results are still positive, namely 69.6%. We hypothesize that this lower predictability is caused by the limited variety of refactoring plans. In fact, we only provide alternatives to move the dependencies down the hierarchy, i.e. to new or existing subtypes, and we neglect the possibility of moving dependencies up the hierarchy.

8.6.3. Combined Results

To provide a conclusion on anti-pattern evaluation, we cross the frequency of occurrence of the anti-patterns (measured the percentage of qualified models containing a given anti-pattern occurrence) with problem-rate (measured by the number of errors divided by number of occurrences) and effectiveness of our rectification plans (quantified as the number of adopted refactoring plans divided by the number of occurrences considered as errors). The higher all these three values are for an anti-pattern, the more useful the anti-pattern is. Anti-patterns that always occur, with a high probability of characterizing a mistake, and with effective associated rectification solutions are more likely to be useful in practice. Less useful anti-patterns, on the other hand, are not those that rarely occur, but instead those that we frequently find but are rarely the source of domain misrepresentations. That is because they require a lot of effort to analyze and provide little gain in model quality. This is not the case of any of the anti-patterns presented in this chapter.

Table 8.8 describes the combined results of the evaluation. Instead of the actual percentages for each measurement, we adopted a discrete scale to classify the values, specified as follows: Very High (80-100%), High (60-80%), Medium (40-60%), Low (20-40%) and Very Low (0-20%).

The results show that among the three anti-patterns presented in this chapter, *RelRig* is the one that can contribute more during model validation. It is frequently found, it often points to a modeling error and we are frequently able to precisely automate its associated refactoring solutions. Although *FreeRole* and *MultDep* did not perform as well, they still fit our criteria for useful anti-patterns, since they suggest a still relevant problem rate and their solutions are highly automatable.

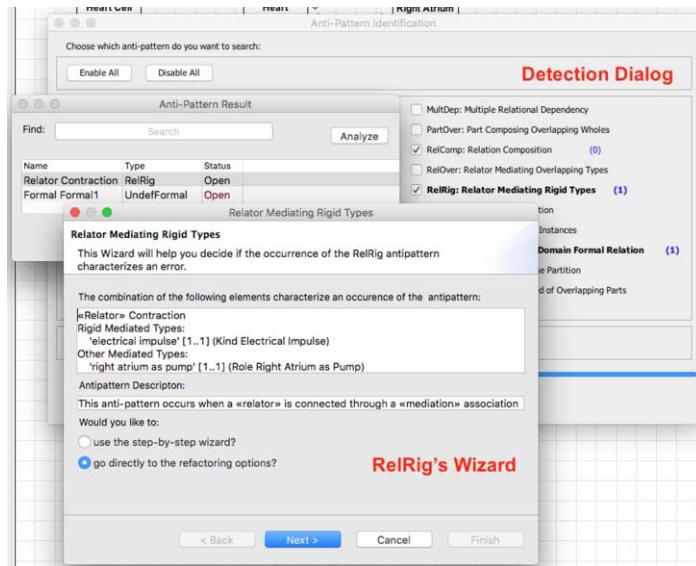


Figure 8.4. Tool support for anti-pattern management implemented in the Menthor Editor.

8.7. Tool Support

Menthor Editor², formerly known as OntoUML Lightweight Editor (OLED), is an ontology-driven conceptual modeling environment. It provides support for model specification (through ontological patterns), automatic syntax verification, validation (through visual simulation) and code generation for the semantic web (RD-F/OWL) and software development (information models). We used the Menthor Editor environment to implement the anti-pattern management functionalities.

In order to make the anti-patterns helpful for model validation, we repeated the strategy adopted in [23]. Firstly, we implemented algorithms to automatically detect occurrences, accessible through the Detection Dialog depicted in Figure 8.4. In the sequence, based on our pre-defined solutions, we implemented wizards to interact with users to support anti-pattern analysis. The wizard for *RelRig* is also depicted in Figure 8.4. Lastly, we implemented algorithms to automatically rectify the model using the input provided during the interaction with the wizard.

8.8. Final Considerations

In this chapter, we extended our work on ontological anti-patterns, proposing three new error-prone structures in combination with pre-defined rectification solutions. In particular, the anti-patterns reported here all related to the notion of roles. Role modeling is of fundamental importance in conceptual modeling in general and in ontology engineering in particular. For instance, in the MGIC

² Available at <http://www.menthor.net>

ontology analyzed in one of our empirical studies, «role» is by far the most used construct in the model with 1066 occurrences. Hence, the identification of these anti-patterns and their associated rectification plans as well as their automation in a model-based computational tool constitutes important contributions to the theory and practice of these disciplines.

In order to identify these anti-patterns, we have used two empirical studies. In the first study, by analyzing a model repository of OntoUML models, we investigated the actual frequency of occurrence of these anti-patterns. In the second study, by using a real-world massive governmental ontology, we have investigated how informative are these anti-patterns (how likely they are to spot an actual modeling error) and how effective are the rectification plans associated to them.

Since anti-patterns signal deviations between intended and valid model instances, and since intended model instances only exist in the mind of domain experts, anti-pattern discovery is a human-centric activity. Hence, the anti-patterns currently making our catalog (reported here and in [23]) were discovered in a heavily manual process. As a future work, we intend to addresses these challenges by studying strategies to (semi) automate the anti-pattern discovery process. For instance, we would like to provide mechanisms that could automatically learn the recurrent correlation between (a) structures in the unintended model instances, (b) structures in the conceptual models that cause them, and (b) solutions provided by the conceptual modelers over (b) in order to rectify the unintended situation identified in (a). Once these strategies are identified and implemented in our computational support, we intend to extend this tool support to be able to automatically identify these anti-patterns across different conceptual models in our model repository. A possibly promising path for investigation in that respect, in the spirit of [1], is the combination of inductive logic learning mechanisms with the counter-example generation capabilities of our model simulation environment (based on Alloy).

Bibliography

- [1] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Automated support for diagnosis and repair. *Communications of the ACM*, 58(2):65–72, 2015.
- [2] J. ao L.R. Moreira, L. Ferreira Pires, M. van Sinderen, and P. Dockhorn Costa. Towards ontology-driven situation-aware disaster management. *Journal of applied ontology*, 10(3-4):339–353, December 2015.
- [3] C. A. M. Bastos, L. Rezende, M. Caldas, A. S. Garcia, S. M. Filho, and J. Castro Junior. Building up a model for management information and knowledge : the case-study for a Brazilian regulatory agency. In *Proceedings of the International Workshop on Software Knowledge*, SKY’11, 2011.
- [4] J. Baumeister and D. Seipel. Anomalies in ontologies with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):55–68, 2010.
- [5] W. Brown, R. Malveau, H. McCormick, and T. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, New York, USA, 1998.

- [6] R. F. Calhau and R. de Almeida Falbo. A configuration management task ontology for semantic integration. In *Proceedings of the 27th Symposium on Applied Computing*, SAC '12, pages 348–353, New York, USA, 2012. ACM.
- [7] H. C. e Silva, R. de Cassia Cordeiro de Castro, M. J. N. Gomes, and A. S. Garcia. Well-founded IT architecture ontology: an approach from a service continuity perspective. In *Proceedings of the 4th Networked Digital Technologies International Conference (NDT'12)*, pages 136–150. Springer, 2012.
- [8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, USA, 1999.
- [9] E. Gamma, R. Johnson, R. Helm, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, USA, 1994.
- [10] N. Guarino and G. Guizzardi. “We need to discuss the relationship”: revisiting relationships as modeling constructs. In *Proceedings of the 27th International Conference on Advanced Information Systems Engineering (CAiSE'15)*, pages 279–294. Springer, 2015.
- [11] G. Guizzardi. *Ontological Foundations for Structural Conceptual Modeling*. Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands, 2005.
- [12] G. Guizzardi. Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In *Proceedings of the 33rd International Conference on Conceptual Modeling (ER'14)*, pages 13–27. Springer, 2014.
- [13] G. Guizzardi and T. P. Sales. Detection, simulation and elimination of semantic anti-patterns in ontology-driven conceptual models. In *Proceedings of the 33rd International Conference on Conceptual Modeling (ER'14)*, pages 363–376. Springer, 2014.
- [14] D. Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.
- [15] A. Koenig. Patterns and antipatterns. *Journal of Object-Oriented Programming*, 8(1):46–48, 1995.
- [16] A. M. Martínez Ferrandis, O. Pastor López, and G. Guizzardi. Applying the principles of an ontology-based approach to a conceptual schema of human genome. In *Proceedings of the 32th International Conference on Conceptual Modeling (ER'13)*, pages 471–478, Berlin, Heidelberg, 2013. Springer.
- [17] J. C. Nardi, R. de Almeida Falbo, J. P. A. Almeida, G. Guizzardi, L. Ferreira Pires, M. J. Van Sinderen, and N. Guarino. Towards a commitment-based reference ontology for services. In *Proceedings of the 17th International Enterprise Distributed Object Computing Conference (EDOC'13)*, pages 175–184. IEEE, 2013.
- [18] OMG. OMG Object Constraint Language (OCL), Version 2.3.1. Technical report, Object Management Group, January 2012.
- [19] M. Poveda-Villalón, M. C. Suárez-Figueroa, and A. Gómez-Pérez. Validating ontologies with OOPS! In *Proceedings of the 18th International Conference Knowledge Engineering and Knowledge Management*, pages 267–281. Springer, 2012.
- [20] C. Roussey, O. Corcho, and L. M. Vilches-Blázquez. A catalogue of OWL

- ontology antipatterns. In *Proceedings of the 5th International Conference on Knowledge Capture*, pages 205–206. ACM, 2009.
- [21] T. P. Sales. *Ontology Validation for Managers*. Universidade Federal do Espírito Santo, Vitória, Brazil, 2014.
 - [22] T. P. Sales, P. P. F. Barcelos, and G. Guizzardi. Identification of semantic anti-patterns in ontology-driven conceptual modeling via visual simulation. In *Proceedings of the 4th International Workshop on Ontology-Driven Information Systems (ODISE'12)*, 2012.
 - [23] T. P. Sales and G. Guizzardi. Ontological anti-patterns: Empirically uncovered error-prone structures in ontology-driven conceptual models. *Data & Knowledge Engineering*, 99:72–104, 2015.

This page intentionally left blank

Chapter 9

Collected Research Questions Concerning Ontology Design Patterns

Karl Hammar, Jönköping University and Linköping University, Sweden

Eva Blomqvist, Linköping University, Sweden

David Carral, Wright State University, Dayton, Ohio, USA

Marieke van Erp, Vrije Universiteit Amsterdam, The Netherlands

Antske Fokkens, Vrije Universiteit Amsterdam, The Netherlands

Aldo Gangemi, CNR-ISTC, Italy

Willem Robert van Hage, Vrije Universiteit Amsterdam, The Netherlands

Pascal Hitzler, Wright State University, Dayton, Ohio, USA

Krzysztof Janowicz, University of California, Santa Barbara, USA

Nazifa Karima, Wright State University, Dayton, Ohio, USA

Adila Krisnadhi, Wright State University, Dayton, Ohio, USA and Universitas Indonesia

Tom Narock, Marymount University, USA

Roxane Segers, Vrije Universiteit Amsterdam, The Netherlands

Monika Solanki, University of Oxford, U.K.

Vojtech Svatek, University of Economics, Prague, Czech Republic

9.1. Introduction

This chapter lists and discusses open challenges for the ontology design pattern (ODP) community in the coming years, both in terms of research questions that will need to be answered, and in terms of tooling and infrastructure that will need to be developed to increase adoption of ODPs in academia and industry. The

chapter is organised into four sections: Section 9.2 focuses on issues pertaining to the patterns themselves, including understanding their features and qualities, and developing pattern languages and standards. Section 9.3 concerns the evaluation and development of methods for using, constructing, and extracting ODPs. Section 9.4 focuses on tooling and infrastructure development, including Ontology Engineering environments that support pattern use, pattern repository development, and sustainability and versioning issues. Finally, Section 9.5 lists some additional research challenges related to pattern-based ontology modularization.

9.2. Patterns and Standards

9.2.1. Quality and Evaluation

Many ontologies are rarely, if ever, reused by third parties. This flies in the face of one of the original main motivations for creating ontologies, namely as shared conceptualizations. However, the exact reasons why ontologies see so little reuse are at this point very poorly understood. Anecdotal evidence indicates that ontologies often are poorly designed and constructed, difficult to understand, insufficiently documented and maintained, too specific or too broad (or both). Social dynamics and findability may also play a role. As ontology design patterns are created and used, it behooves the community to really understand the causes for the lack of reuse.

Without quality documentation, it is nearly impossible for ontology engineers to discover existing reusable ODPs, whether by browsing manually or using some search tool. Thus understanding how to best document ODPs (which documentation components are needed, in which order should they be displayed, using which graphical layout, etc) and developing benchmarks for evaluating ODP documentation are important research directions.

Further, the ODP community needs to address possible causes related to poor design, to understandability issues, and to finding an appropriate balance for specificity versus generality. A thorough and evidence-supported understanding of these issues, e.g. what exactly “poor design” and “good design” are, appears to be necessary to advance on these fronts; investigations into ontological anti-patterns fall into this realm [19, 20]. In particular, there seems to be a lack of user-centric evaluations addressing the ontology reuse issue, with only a few notable exceptions, e.g. [10].

In addition to the above, even if a greater understanding of ODP design and documentation are developed, we still need to challenge and evaluate our basic assumptions. We often take as a given that properly supported, ODP use will foster reuse of ontologies, help clean up messy data and facilitate data integration, when in fact, these claims are to a large degree still unverified hypotheses. An important step forward for the ODP community would be to show the validity of such claims. This could be done via empirical studies such as [10] and the development and proper documentation of useful applications which make use of the ODP paradigm.

9.2.2. Pattern Languages and Standards

Currently, ontology design patterns are mostly presented in the form of OWL files, with some minimalistic accompanying documentation. While OWL can indeed be used to express the core of a pattern, it does not provide native ways to represent additional information which would be helpful for reuse or organization of patterns.

For example, it would seem to be important to understand in depth (and be able to represent formally) how different patterns relate to each other. This is important to understand compositionality aspects, e.g. how to create an ontology based on existing ontology design patterns. Key relations are specialization, generalization, and composition and are discussed as in [17], but often relationships are more complex. For example, there are notions of views or shortcuts, see [9, 12], which can be understood as a type of temporary simplification of a pattern.

Using OWL to express axiomatizations provided with patterns is also limiting because the open-world assumption underlying OWL does not cater for expressing non-monotonic constructs like constraints, and because some patterns call for an axiomatization of, say, transitive closure or general rules not expressible in OWL DL. Some first steps towards developing such a representation language have already been undertaken [5, 7] but more work remains to be done.

Another challenge concerns naming in ODPs – currently, the lexical aspect of patterns is rather ad hoc. There has already been some work specifically devoted to ontological naming conventions [21] and even mentioning cross-entity structures [23], but little long-term systematic work has been carried out, and the respective category of the ODP portal is, consequently, empty.¹ The naming aspect is important both for content patterns (which are a kind of mini-ontologies with their own vocabulary) and for logical patterns, where structural transformation from one modeling variant (e.g., reification or meta-modeling) may also involve transformation of naming such as generation of derived word forms of different parts of speech.

9.3. Method Development

9.3.1. Usage Methods

Currently there is only one published ontology engineering methodology that explicitly mentions the use of ODPs: the eXtreme Design (XD) methodology [1, 14, 15]. If we compare this situation with that in software engineering, we find that in that discipline, almost regardless of what methodology you use, you will take design patterns into account in some way or another. Following this example, in ontology engineering it would probably be beneficial to be able to use ODPs in any ontology engineering methodology, and not only in XD. This raises the question of what ODP usage would look like in other kinds of ontology

¹<http://ontologydesignpatterns.org/wiki/Category:NamingOP>

engineering methodologies, and if different variations of the ODP concept may be needed there.

We also note that there are still open questions relating to the improvement and further development of the XD methodology. Improvement suggestions include providing ways of tailoring the methodology to different project settings such as the staffing situation, duration of the project, the overall goal of the project (e.g. type of ontology, how much focus is put on correct concept definitions and terminology vs. logical structure and reasoning capabilities), and also detailing existing steps of the methodology to better fit realistic project settings. The latter includes the question of reuse – how do you reuse an existing knowledge structure (potentially an existing ontology) and extend that by using ODPs? How do you know an ODP fits your existing structure, so that you know what ODPs to select for your extensions?

9.3.2. ODP Development

The advance of ontology design patterns is caught in a catch-22: In order to provide convincing evidence for the added value of ODPs, the community requires access to a well-organized, well-documented, and well-maintained set of inter-linked high-quality ontology design patterns. At the same time, however, there is a lack of incentive (and funding) to provide these, as long as this added value has not yet been convincingly demonstrated.

A joint effort to create, document, and properly catalogue key ontology design patterns will be needed, as well as a discussion on which patterns are needed and how they should be provided. A pattern creation effort is ongoing for the geosciences domain, in form of the U.S. GeoVocamps [9], but the process is still very ad-hoc, and a broader organized effort will be needed.

Good, and highly visible, use cases can be powerful drivers of such community efforts. While use cases for pattern-driven ontology engineering exist (e.g., the recent [11, 13]), they need to be catalogued, made accessible, and assessed regarding the relevance of ODPs to the scenario. Potential additional use cases could be developed with focus on data publishing and reuse, e.g. in the realm of linked data [18], which has seen much work in recent years. A regular challenge or competition event at the WOP workshops could be set up to facilitate this development.

One recent area where the development of more ODPs could fill a clear need would be when applying some non-standard features in ontologies. There are many tutorials, courses, modelling guidelines and ODPs that can help when using standard OWL. However, every year new extensions to the standards are developed, e.g. in terms of managing and reasoning over streaming data, in terms of modelling and reasoning over uncertainty etc. Nevertheless, there are so far few ODPs for such “extensions” to standard OWL, although this is an area where help and support for the ontology engineer would be really badly needed.

Another challenge concerns methods and best practices for the development of new patterns, which is currently undertaken in a rather ad-hoc manner. Just as guidelines are available for publishing ontologies and linked data, best practices must be developed that guide pattern developers in a systematic way to address

the problem of developing and publishing a design pattern. These guidelines must empower the pattern developers to ascertain when their pattern axiomatisation is indeed a pattern and when it moves into the realm of a full fledged ontology before they publish it. Best practices may include defining the scope of the pattern, the minimum number of use cases that need to be made available as examples of the pattern application, the level and kind of axiomatisation that is appropriate and the possible extension points for future exploitation of the pattern amongst others, etc. The inclusion of patterns in the ontologydesignpatterns.org patterns repository should be governed by validating these criteria.

In addition to traditional OWL-based ODPs, we should consider that ontology patterns (especially content/knowledge patterns) exist in more or less explicit forms in datasets and models developed within diverse disciplines beyond ontology engineering and the Semantic Web, including formal and computational linguistics, lexical resources, data modelling, stylesheets, microformats, web pages, HCI, tables, etc. Making these implicit patterns explicit, and finding links, is a big challenge, which we have just started. Cf. [6] for a related manifesto. In particular, the patterns emerging from NLP and linguistic resources, and those from ontology engineering and the Semantic Web have important relations, in terms of grounding, empirical evidence, cognitive relevance, etc. (cf. the NLP formal integration work enabled by FRED [16], and the correspondences found in FrameBase).

9.3.3. ODP Extraction

The development of Software Engineering design patterns has conventionally followed a bottom-up approach where recurring patterns are extracted from existing software. This also serves as a validation of the usage of the pattern. However, the development of ODPs has largely happened in a top-down manner. This was justified in the early days of ontology development due to a lack of critical mass. However, strategies are now required that encourage bottom-up development and enable automated extraction of patterns from a corpus of ontologies.

Such a corpus, that is as of yet relatively untapped, in fact exists: many well known and widely used ontologies were developed before ODPs were introduced. Some of these ontologies are upper level such as BFO and SUMO, while others such as SNOMED-CT and CIDOC-CRM are relatively large. These ontologies potentially encode a number of ODPs, a fact which is implicitly validated by their widespread usage. Analyzing and restructuring these ontologies to identify patterns using traditional manual methods is a non-trivial task requiring significant support, which is often not available. The development of automated extraction strategies and tools could however uncover many currently hidden ODPs, thereby bridging the gap between the demand and availability of ODPs, and overcoming the catch-22 discussed in Section 9.3.2.

9.4. Tooling and Infrastructure

9.4.1. Pattern-Capable Ontology IDE:s

Which kind of tool support that is needed and best suited for ODP development and use, is an open question. Some tools have been proposed (such as plugins for the NeOn toolkit, and more recently for WebProtégé [8]) but they do not cover all possible tasks where ODPs can be applied, and in particular there are currently no tools supporting the construction or extraction of patterns, and very few that support sharing, discussing and collecting them – while some online portals do exist, there are no actual client-side tools.

Also, the tooling that does exist is very traditional, introducing the ODPs as a small variation of the normal workflow. An interesting question is whether there are any opportunities for more disruptive changes in tooling that could be supported by ODPs. For instance, (content) ODPs are sometimes viewed as small puzzle pieces, but so far there have not been any attempts to create tools that truly operate on the level of ODPs as their primitives (rather than OWL languages primitives). One could imagine taking ontology engineering to an entirely new level, where the underlying logical language is hidden behind a pattern language.²

Additionally it is important for the modeler to become aware of the availability of a suitable pattern when trying to model a certain state of affairs – currently there is a lack of pattern discovery services in general, as observed in [2] as one of the ODP adoption barriers. While keyword search may be of some use for content patterns it is insufficient for logical-structural patterns. The proposed development of an expressive pattern representation language is important but does not suffice by itself if modeling situations are not described as well. Ideally, the retrieval of patterns should be closely interconnected with the support of its reuse. When the modeling challenge mainly consists of expressivity constraints of the target language (e.g., related to n-ary relationships or meta-typing), the modeled situation can possibly be described using first-order logic or a dedicated language such as PURO [22], and starting from this representation, the reuse of the pattern might be invoked semi-automatically for the target ontology [4]. If the problem is of more complex nature, some kind of interactive navigation over some “ontology of ontological modeling problems” to a node referencing adequate patterns might suit better, and the reuse itself is then likely to be more manual as well. In any case, such supportive technologies are currently either missing or immature.

9.4.2. Repository Development

Ontology patterns can be needed in many different lexical and structural contexts. An advanced semantic search capability would boost their usage, but e.g. linking competency questions to potentially useful patterns is not trivial, especially if the structure of the question has to be taken into account. There is room here to join in with research on question-answering on linked data.

²See also the discussion on pattern language in Section 9.2.2.

9.4.3. Sustainability Challenges

The provision, maintenance and documentation of ontology design patterns are currently done in a very ad-hoc manner. While the community has a rudimentary portal,³ it lends only limited support and structure. Versioning aspects and other software engineering related issues need to be addressed, and better tool support for pattern-driven ontology engineering needs to be developed. Sustainability related research questions are about finding out how to best implement infrastructure and apply structured and qualitative software engineering principles to ODPs. What would a tool for the efficient support of ODP-driven ontology-engineering look like? How should documentation be provided? What does versioning mean for ontologies or ODPs on the Web?

9.5. Challenges Related to Pattern-Based Ontology Modularization

Usually, ontology design patterns are merely thought of as building blocks for traditional ontologies. However, it can be argued that there may also be added value in using ODPs for other than this very specific purpose.

One example would be the possible modularization of ontologies [11]. The idea, in this case, would be to somehow preserve the underlying pattern structure, in the form of connected modules, as part of the final ontology. The potential advantages of this appear to be rather self-evident: Modularization would simplify documentation, understanding, and thus reuse. At the same time, however, it would also make it easier to modify parts of an ontology, in particular if such a part be contained within a specific module. Furthermore, repurposing of an ontology may be much easier if modularized, because replacement of modules may be a more straightforward means of adapting an ontology to a new purpose.

Current ontology languages, however do not provide suitable means for representing such modular ontologies and their dependencies. There is also little experience in the ramifications arising from module updates, and of the role of axiomatisations with respect to modularization and module modification. We also do not know about concrete studies which would verify the more obvious promises of such modularization.

Explicit preservation of ODPs used for ontology construction (e.g., as modules) also have the potential to simplify ontology engineering tasks which usually come later in the development pipeline. For example, a systematic reuse of ODPs in ontology construction should rather obviously make ontology alignment a much more tangible task, and may be a practical path forward to overcoming the current bottlenecks in that research area [3]. In fact, it is conceivable that popular and often-reused ODPs could be enhanced with additional information tailored towards making alignments even simpler. Such information could, for example, take the form of annotated natural language sentences which act as linguistic samples for pattern content, and which thus can serve as hooks for linguistic techniques to be used for alignment. Such information could also be used to

³<http://ontologydesignpatterns.org>

enhance ontology population systems, in particular with respect to ontology population from texts. Systematic studies to verify these promises and which would show how to realize such systems have not been pursued so far, to the best of our knowledge.

Acknowledgements

Some of the authors acknowledge partial support by the National Science Foundation through the project *EarthCube Building Blocks: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*. Parts of this chapter appeared already in [2].

Bibliography

- [1] E. Blomqvist, K. Hammar, and V. Presutti. Engineering ontologies with patterns – the eXtreme design methodology. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016.
- [2] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2015.
- [3] M. Cheatham and P. Hitzler. String similarity metrics for ontology alignment. In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web – ISWC 2013 – 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21–25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*, pages 294–309. Springer, 2013.
- [4] M. Dudáš, T. Hanzal, V. Svátek, and O. Zamazal. Obowlmorph: Starting ontology development from puro background models. To be published via Springer LNCS.
- [5] R. Falbo, M. Barcellos, F. Ruy, G. Guizzardi, and R. Guizzardi. Ontology pattern languages. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016.
- [6] A. Gangemi and V. Presutti. Towards a pattern science for the semantic web. *Semantic Web*, 1(1, 2):61–68, 2010.
- [7] G. Guizzardi. Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In E. S. K. Yu, G. Dobbie, M. Jarke, and S. Purao, editors, *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27–29, 2014. Proceedings*, volume 8824 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2014.
- [8] K. Hammar. Ontology design patterns in webprotege. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web*

- Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015., volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [9] P. Hitzler, K. Janowicz, and A. A. Krisnadhi. Ontology modeling with domain experts: The GeoVocamp experience. In C. d'Amato, F. Lécué, R. Mutharaju, T. Narock, and F. Wirth, editors, *Proceedings of the 1st International Diversity++ Workshop co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015.*, volume 1501 of *CEUR Workshop Proceedings*, pages 31–36. CEUR-WS.org, 2015.
 - [10] C. M. Keet. The use of foundational ontologies in ontology development: An empirical assessment. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 321–335. Springer, 2011.
 - [11] A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. A. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. W. Finin, P. Ji, M. B. Jones, N. Karima, K. A. Lehnert, A. Mickle, T. W. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The GeoLink modular oceanography ontology. In M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 2015.
 - [12] A. A. Krisnadhi. *Ontology Pattern-Based Data Integration*. PhD thesis, Wright State University, 2015.
 - [13] A. A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. A. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. Finin, P. Ji, M. B. Jones, N. Karima, K. A. Lehnert, A. Mickle, T. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The geolink framework for pattern-based linked data integration. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015.*, volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
 - [14] V. Presutti, E. Blomqvist, E. Daga, and A. Gangemi. Pattern-based ontology design. In M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World.*, pages 35–64. Springer, 2012.
 - [15] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. extreme design with content ontology design patterns. In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svátek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009) , collocated with the 8th International Semantic Web Conference (ISWC-2009) , Washington D.C., USA, 25 October, 2009.*, volume 516 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

- [16] V. Presutti, F. Draicchio, and A. Gangemi. Knowledge extraction based on discourse representation theory and linguistic frames. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Aquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, volume 7603 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2012.
- [17] V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In Q. Li, S. Spaccapietra, E. S. K. Yu, and A. Olivé, editors, *Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings*, volume 5231 of *Lecture Notes in Computer Science*, pages 128–141. Springer, 2008.
- [18] V. Rodríguez-Doncel, A. A. Krisnadhi, P. Hitzler, M. Cheatham, N. Karima, and R. Amini. Pattern-based linked data publication: The linked chess dataset case. In O. Hartig, J. Sequeda, and A. Hogan, editors, *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, US, October 12th, 2015.*, volume 1426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [19] T. Sales and G. Guizzardi. Ontological anti-patterns. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016.
- [20] T. P. Sales and G. Guizzardi. Ontological anti-patterns: Empirically uncovered error-prone structures in ontology-driven conceptual models. *Data & Knowledge Engineering*, 99:72–104, 2015.
- [21] D. Schober, I. Tudose, V. Svátek, and M. Boeker. OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4. *Journal of Biomedical Semantics*, 3(2):1, 2012.
- [22] V. Svátek, M. Homola, J. Kluka, and M. Vacura. Mapping structural design patterns in OWL to ontological background models. In V. R. Benjamins, M. d'Aquin, and A. Gordon, editors, *Proceedings of the 7th International Conference on Knowledge Capture, K-CAP 2013, Banff, Canada, June 23-26, 2013*, pages 117–120. ACM, 2013.
- [23] V. Svátek, O. Sváb-Zamazal, and V. Presutti. Ontology naming pattern sauce for (human and computer) gourmets. In E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svátek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009) , collocated with the 8th International Semantic Web Conference (ISWC-2009) , Washington D.C., USA, 25 October, 2009.*, volume 516 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

Part II

Ontology Design Patterns In Practice

This page intentionally left blank

Chapter 10

Ontology Design Patterns for Linked Data Publishing

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

Nazifa Karima, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Reihaneh Amini, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Víctor Rodríguez-Doncel, Ontology Engineering Group, Universidad Politécnica de Madrid, Spain

Krzysztof Janowicz, STKO Laboratory, University of California, Santa Barbara, CA, USA

10.1. Introduction

In Chapter 1 [17], we have given a worked example on how to develop a modular ontology using ontology design patterns. In the following, we will work through an example process how to use such an ontology for publishing Linked Data. Supplementary material, such as additional examples, OWL, and RDF files, are available at <http://dase.cs.wright.edu/content/pattern-driven-linked-data-publishing-primer>.

Linked Data – or Linked *Open* Data if the reference is to Linked Data that is openly available in the sense of liberal re-use licenses [25] – refers to the publishing of data on the Web following a few very straightforward principles [7], foremost among them the use of the Resource Description Framework (RDF).¹ Linked Data has seen very significant uptake since its inception in 2006.² In 2014, over

¹<https://www.w3.org/RDF/>

²<http://lod-cloud.net/>

1,000 linked datasets were counted [28], a more than threefold increase over 2011. Also, the LOD Laundromat [4, 24] website³ currently reports over 650,000 RDF documents totaling over 38,600,000,000 triples.

The benefits of Linked Data are rather obvious and well publicized [12]: simply having large amounts of data on various subjects available in structured form using a standardized format like RDF makes it easier to find, ingest, reuse this data by third parties for all kinds of purposes. Some prominent examples include the performance improvements to the IBM Watson system due to ingesting the DBpedia linked dataset [10], and the use of Linked Data at the nytimes.com portal⁴ and the bbc.com portal. The latter started with the deployment of the BBC Dynamic Semantic Publishing for the 2010 World Cup content [23], which was reused for the London 2012 Olympics content [3], and later on extended to the whole range of content that may be of interest to the BBC [2].

However, reuse of Linked Data has not taken off at large scale yet, and some of the reasons for this can be traced back to the often significant effort needed in finding, understanding and assessing such datasets, and in curating them to make them fit for the task at hand [26]. In our experience, some⁵ of the significant cost factors arise out of the neglect of quality metadata aspects, such as the failure to provide an ontology underlying the data graph structure; graph structure (and underlying ontology) not being modeled according to best practices as they arise, e.g., out of ontology design patterns work; large, monolithic graphs and ontologies which are very hard to inspect and understand by humans; and so forth [15, 26].

Consequently, we advocate that linked data publishing should be done in accordance with a high-quality ontology to convey, explain, and enrich the bare linked data graph. In the sequel, we assume that such an ontology has already been developed based on ontology design pattern principles. This ontology provides us with definition of vocabulary terms that will be used to annotate the data and essentially act as the schema for the data. To make things more concrete, we shall describe the thought processes and an example workflow leading to the publishing of a linked dataset with vocabulary obtained from such an ontology. For the example workflow, we shall build on the ontology developed and described in Chapter 1 [17], make several adjustments to it and tie some loose ends to make it more suitable for linked data publishing. Afterwards, we will illustrate the steps one needs to take in setting up an appropriate infrastructure to publish linked data based on the ontology so that the data is published according to the Linked Data principles [7].

10.2. Vocabulary Preparation

Our starting point is Fig. 1.9 of Chapter 1 [17], which refers to several external patterns. We now need to make precise decisions how to resolve these external references, i.e., whether to indeed import some external model, or to use a stub, or use a simplified solution for the time being. In addition, we modify the axioms

³<http://lodlaundromat.org/> – retrieved 24 February 2016

⁴As reported by Evan Sandhaus in his keynote talk at the 9th International Semantic Web Conference (ISWC 2010) in Shanghai, China, November 2010.

⁵But certainly not all, see e.g. [30].

from Fig. 1.12 and 1.13 in accordance to the modeling decision we are about to make below. In making these changes, however, we generally remain truthful to the original model.

- We retain the use of stubs for Chess Tournament, Chess Opening, and Chess Game Result as indicated. That is, we shall have possibly blank nodes, which are simply typed as indicated. For the Chess Opening stub, we remove `hasName` property and consider the ECO code sufficient for our purpose.
- We use a similar stub for Agent and Place i.e. attaching a string as name using the `hasName` property.
- We omit axioms involving the `startsAtTime` and `endsAtTime` property of Agent Role.
- We remove Event because it was used for design, and there seems no added value in producing the additional triples for instantiating it at this stage.
- With Event removed, the inherited information regarding Place is attached to ChessGame directly, and additionally, in place of TemporalExtent, we also attach `xsd:dateTime`⁶ to ChessGame. This is of course an oversimplification because several games may be on the same day, and the order may matter. However, in practice simplifications are sometimes needed to go forward; we acknowledge that this particular simplification may complicate the integration with more fine-grained data, and may preclude some uses that rely on a more fine-grained modeling. PGN files, which will be the data source we will currently only look at, provide the location and dates for chess games.
- Similarly, we attach `xsd:dateTime` through `atTime` and Place through `atPlace` to ChessTournament, which was originally also modeled as event.
- We remove the more general scoped domain and range restrictions for `subEventOf` property (with Event as both domain and range of this property). The axioms that express domain and range restrictions for this property where the domain and range pairs are ChessGame and ChessTournament, as well as HalfMove and ChessGame will be retained.
- We remove HalfMoveAnnotation for the time being, mainly because republishing of comments from PGN files may be difficult in terms of copyright; the move sequence itself is unproblematic, see [29].
- We remove originatesFrom for the time being, because we will look only at PGN files as sources for data at this stage.

The resulting graph is depicted in Fig. 10.1, while the modified set of axioms can be found in Fig. 10.2.

10.3. Views and Shortcuts

An expressive schema such as the one in Fig. 10.1 is extremely helpful to retain the benefits of high-quality ontology modeling for linked data publishing [8], including ease of understanding of the graph structure; ease of schema and thus graph

⁶The prefix `xsd` stands for <http://www.w3.org/2001/XMLSchema#>

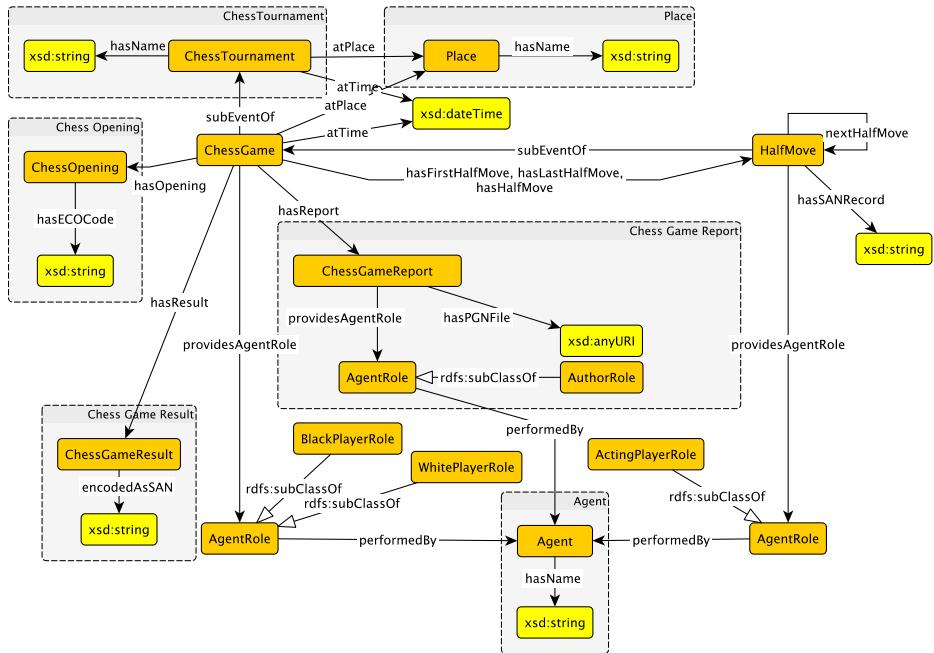


Figure 10.1. Chess Ontology modified from Fig. 1.9 of Chapter 1 [17].

extension without violating past modeling or requiring major modifications to past modeling; ease of reuse because the graph structure is easy to understand and modify; ease of integration with other data since the schema – and thus the RDF graph structure – lends itself more easily to such integration; ease of reuse of part of the RDF graph, since the schema – and thus the RDF graph – is already modularized.

However, the expressive schema also comes with a number of drawbacks: the graph structures are more complicated, and thus it takes more time and effort to cast data into it, they are more difficult to understand, and for those intending to re-use the data, they have to deal with a much more complicated schema than their specific use case requires. Furthermore, the resulting RDF graphs will usually be much larger than they would be if a much simpler schema were used.

Of course, the advantages of a complex schema are the disadvantages of a simple schema and vice versa. This raises the question about the best choice in any given situation, or the question about how to find a good compromise.

However, as we will argue and show below, we can also instead reap the best of both worlds, by moving to a *dual* schema, consisting of both a complex and one (or several) compatible simplified versions. In this case, users can choose the perspective which fits them, and even switch seamlessly between them if this is desired. We will also see below that the overhead resulting from such a dual schema is reasonable.

The idea is as follows: Once a complex schema like the one depicted in

$\text{AgentRole} \sqsubseteq (=1 \text{ performedBy.Agent}) \sqcap \forall \text{performedBy.Agent}$	(10.1)
$\exists \text{performedBy.Agent} \sqsubseteq \text{AgentRole}$	(10.2)
$T \sqsubseteq \forall \text{pAR.AgentRole}$	(10.3)
$\text{ChessGame} \sqsubseteq \exists \text{atPlace.Place} \sqcap \forall \text{atPlace.Place}$	(10.4)
$\text{ChessGame} \sqsubseteq \exists \text{atTime.xsd:dateTime} \sqcap \forall \text{atTime.xsd:dateTime}$	(10.5)
$\text{ChessGame} \sqsubseteq \exists \text{pAR.BlackPlayerRole} \sqcap \exists \text{pAR.WhitePlayerRole}$	(10.6)
$\exists \text{subEventOf.ChessTournament} \sqcup \exists \text{hasOpening.ChessOpening} \sqsubseteq \text{ChessGame}$	(10.7)
$\exists \text{hasResult.ChessGameResult} \sqcup \exists \text{hasReport.ChessGameReport} \sqsubseteq \text{ChessGame}$	(10.8)
$\text{ChessGame} \sqsubseteq \forall \text{subEventOf.ChessTournament} \sqcap \forall \text{hasOpening.ChessOpening}$	(10.9)
$\text{ChessGame} \sqsubseteq \forall \text{hasResult.ChessGameResult} \sqcap \forall \text{hasReport.ChessGameReport}$	(10.10)
$\text{BlackPlayerRole} \sqcup \text{WhitePlayerRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^- \cdot \text{ChessGame})$	(10.11)
$\text{ChessGame} \sqsubseteq (=1 \text{ hasFirstHalfMove.HalfMove}) \sqcap (=1 \text{ hasLastHalfMove.HalfMove})$	(10.12)
$\text{ChessGame} \sqsubseteq (=1 \text{ hasLastHalfMove.HalfMove})$	(10.13)
$\text{hasHalfMove} \sqsubseteq \text{subEventOf}^-$	(10.14)
$\text{hasFirstHalfMove} \sqsubseteq \text{hasHalfMove}$	(10.15)
$\text{hasLastHalfMove} \sqsubseteq \text{hasHalfMove}$	(10.16)
$\text{HalfMove} \sqsubseteq \text{Event} \sqcap \exists \text{pAR.ActingPlayerRole} \sqcap (=1 \text{ hasHalfMove}^- \cdot \text{ChessGame})$	(10.17)
$\text{ActingPlayerRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^- \cdot \text{HalfMove})$	(10.18)
$\text{HalfMove} \sqsubseteq (\leq 1 \text{ nextHalfMove.HalfMove}) \sqcap \neg \exists \text{nextHalfMove.Self}$	(10.19)
$\exists \text{subEventOf.ChessGame} \sqcup \exists \text{nextHalfMove.HalfMove} \sqsubseteq \text{HalfMove}$	(10.20)
$\exists \text{hasSANRecord.xsd:string} \sqsubseteq \text{HalfMove}$	(10.21)
$\text{HalfMove} \sqsubseteq \forall \text{subEventOf.ChessGame} \sqcap \forall \text{nextHalfMove.HalfMove}$	(10.22)
$\text{HalfMove} \sqsubseteq \forall \text{hasSANRecord.xsd:string}$	(10.23)
$\text{ChessTournament} \sqsubseteq \exists \text{atPlace.Place} \sqcap \forall \text{atPlace.Place}$	(10.24)
$\text{ChessTournament} \sqsubseteq \exists \text{atTime.xsd:dateTime} \sqcap \forall \text{atTime.xsd:dateTime}$	(10.25)
$\text{ChessTournament} \sqcup \text{Place} \sqcup \text{Agent} \sqsubseteq \forall \text{hasName.xsd:string}$	(10.26)
$\exists \text{hasECOCode.xsd:string} \sqsubseteq \text{ChessOpening}$	(10.27)
$\text{ChessOpening} \sqsubseteq \forall \text{hasECOCode.xsd:string}$	(10.28)
$\exists \text{encodedAsSAN.xsd:string} \sqsubseteq \text{ChessGameResult}$	(10.29)
$\text{ChessGameResult} \sqsubseteq \forall \text{encodedAsSAN.xsd:string}$	(10.30)
$\text{ChessGameReport} \sqsubseteq \exists \text{pAR.AuthorRole}$	(10.31)
$\text{AuthorRole} \sqsubseteq \text{AgentRole} \sqcap (=1 \text{ pAR}^- \cdot \text{ChessGameReport})$	(10.32)
$\exists \text{hasPGNFile.xsd:anyURI} \sqsubseteq \text{ChessGameReport}$	(10.33)
$\text{ChessGameReport} \sqsubseteq \forall \text{hasPGNFile.xsd:anyURI}$	(10.34)
$\text{DisjointClasses}(\text{AgentRole}, \text{Agent}, \text{ChessGame}, \text{ChessTournament}, \text{HalfMove}, \text{Place},$	(10.35)
$\text{ChessOpening}, \text{ChessGameResult}, \text{ChessGameReport})$	
$\text{DisjointClasses}(\text{BlackPlayerRole}, \text{WhitePlayerRole}, \text{ActingPlayerRole}, \text{AuthorRole})$	(10.36)

Figure 10.2. Chess Game axioms after simplifying Fig. 1.12 and 1.13 according to Section 10.2; pAR stands for `providesAgentRole`.

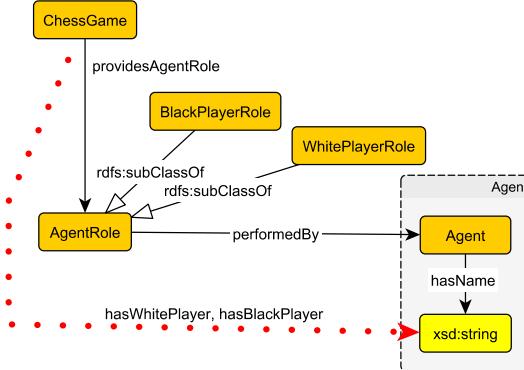


Figure 10.3. Example for a shortcut.

Fig. 10.1 has been developed, a simplified *view* can be established by way of so-called *shortcuts* through the graph [16]. Compilation of data from the complex schema to the simplified view is straightforward using rules, as we will see below.

We first give a small example before providing a complete simplified view for our chess ontology. Consider Fig. 10.3, which shows the part of the ontology connecting *ChessGame* to a player's name; the player could be the white or the black player. The dotted red arrow in this graph indicates two shortcuts, which we name *hasWhitePlayer*, for the *WhitePlayerRole*, and *hasBlackPlayer* for the *BlackPlayerRole*.

Given a populated ontology, we can then materialize the two shortcuts, by firing the following rules – pAR is used as abbreviation for *providesAgentRole*.

$$\begin{aligned}
 & \text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{WhitePlayerRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasWhitePlayer}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{BlackPlayerRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasBlackPlayer}(x, s)
 \end{aligned}$$

These rules – like most rules in fact [18] – can also be expressed in description logics, and thus can often also be expressed in OWL DL [20]. We just give an example for the first rule and refer the interested reader to [18, 20] for further details on how to do this transformation. The main idea is that we associate each class name in the rule with a property that is not previously occurring in the ontology. In the case of the first rule above, we associate *ChessGame* with R_1 , *WhitePlayerRole* with R_2 , and *Agent* with R_3 where all R_1 , R_2 , and R_3 are freshly introduced. The rule then becomes the following set of axioms and the properties R_1 , R_2 , and R_3 are called the *rolifications* of *ChessGame*, *WhitePlayerRole*, and

Agent, respectively [18].

$$\begin{aligned}
 \text{ChessGame} &\sqsubseteq \exists R_1.\text{Self} \\
 \text{WhitePlayerRole} &\sqsubseteq \exists R_2.\text{Self} \\
 \text{Agent} &\sqsubseteq \exists R_3.\text{Self} \\
 R_1 \circ \text{pAR} \circ R_2 \circ \text{performedBy} \circ R_3 \circ \text{hasName} &\sqsubseteq \text{hasWhitePlayer}
 \end{aligned}$$

In this particular case, though, the property chain in the last of these axioms cannot be converted into OWL DL, since `hasName` is a datatype property. This limitation of OWL was discussed in [19].

Mapping in the other direction, from the simplified view to the complex schema, is not as straightforward. First of all, even expressing the bare logical bones of this transformation in a knowledge representation language used in web ontologies is tricky, see the discussion in [19]. Furthermore, the transformation necessitates either generating blank nodes or, if the newly introduced nodes need to be named, minting new identifiers. In the aforementioned rules, the new nodes correspond to the variable y and z . Even more complicated is the fact that co-reference resolution may need to be performed on some of these new nodes, e.g., on the instances of `Agent`, since they may identify some entity occurring elsewhere in the dataset. That this direction is non-trivial indicates again that there is added value – more semantics – in the expressive schema.

The transformation can obviously be done using software specially written for this purpose, typically by making use of RDF APIs, some of which are listed at the end of Section 10.6.1.1. Alternatively, one could express the blank node generation and URI minting inline within a SPARQL query [11] with the help of a few SPARQL built-in functions and some naming convention. This way of expressing the transformation using SPARQL shall be briefly explained in Section 10.6.2

The complete set of shortcuts for our ontology are indicated by dotted arrows in Fig. 10.4. The corresponding rules are given in Fig. 10.5. The complete simplified view is depicted in Fig. 10.6. We are now ready to get down to a more concrete implementation work to publish linked data based on the Chess ontology pattern.

10.4. URI Minting

In Linked Data context, each term in linked datasets and ontologies are called *resources*. To conform with the Linked Data principles [7], we need to make all such resources Web-dereferenceable by assigning a *Uniform Resource Identifier* (URI) to each of them. Some resources may have been defined externally in another linked dataset, which is not under our control. For them, a URI is typically already provided and we just need to use it. For the rest, however, we need to *mint* appropriate URIs by following the design principles below [14].

1. Every resource, aside from literal values, should be assigned a HTPP URI as its identifier.⁷

⁷One can use the plain HTTP URI, i.e., with `http:` prefix, or HTTPS, i.e., with `https:`

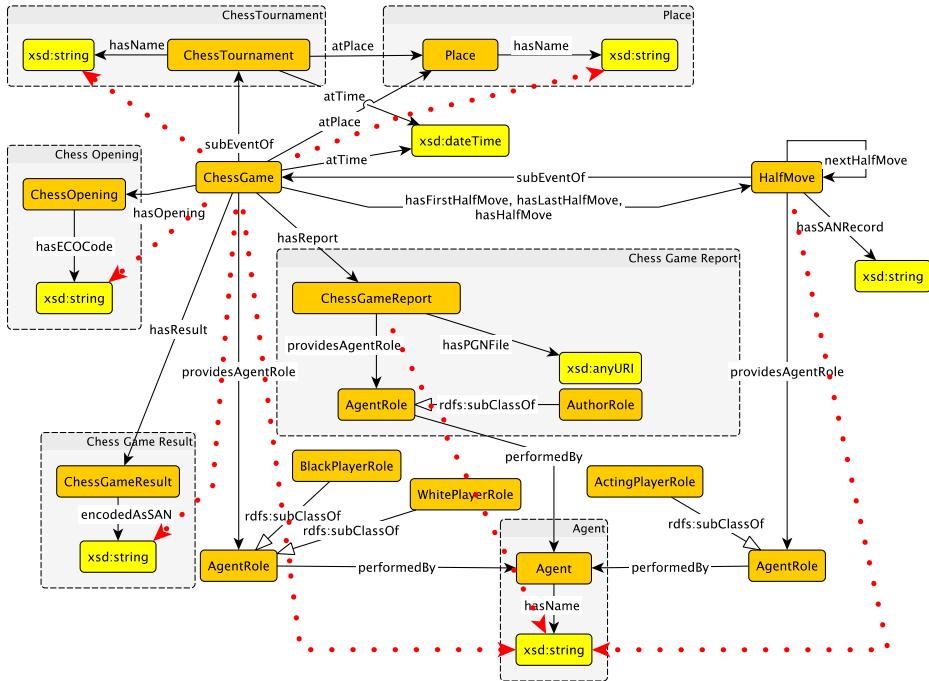


Figure 10.4. Shortcuts indicated by dotted arrows.

$$\begin{aligned}
 & \text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{WhitePlayerRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasWhitePlayer}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{BlackPlayerRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasBlackPlayer}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{hasResult}(x, y) \wedge \text{ChessGameResult}(y) \wedge \text{encodedAsSAN}(y, s) \\
 & \quad \rightarrow \text{hasResultSAN}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{hasOpening}(x, y) \wedge \text{ChessOpening}(y) \wedge \text{hasECOCode}(y, s) \\
 & \quad \rightarrow \text{hasOpeningECO}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{subEventOf}(x, y) \wedge \text{ChessTournament}(y) \wedge \text{hasName}(y, s) \\
 & \quad \rightarrow \text{atChessTournament}(x, s) \\
 & \text{ChessGame}(x) \wedge \text{subEventOf}(x, y) \wedge \text{ChessTournament}(y) \wedge \text{atPlace}(y, z) \\
 & \quad \wedge \text{Place}(z) \wedge \text{hasName}(z, s) \rightarrow \text{atPlaceNamed}(x, s) \\
 & \text{ChessGameReport}(x) \wedge \text{pAR}(x, y) \wedge \text{AuthorRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasAuthor}(x, s) \\
 & \text{HalfMove}(x) \wedge \text{pAR}(x, y) \wedge \text{ActingPlayerRole}(y) \wedge \text{performedBy}(y, z) \\
 & \quad \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{playedBy}(x, s)
 \end{aligned}$$

Figure 10.5. Shortcut rules leading to our simplified view.

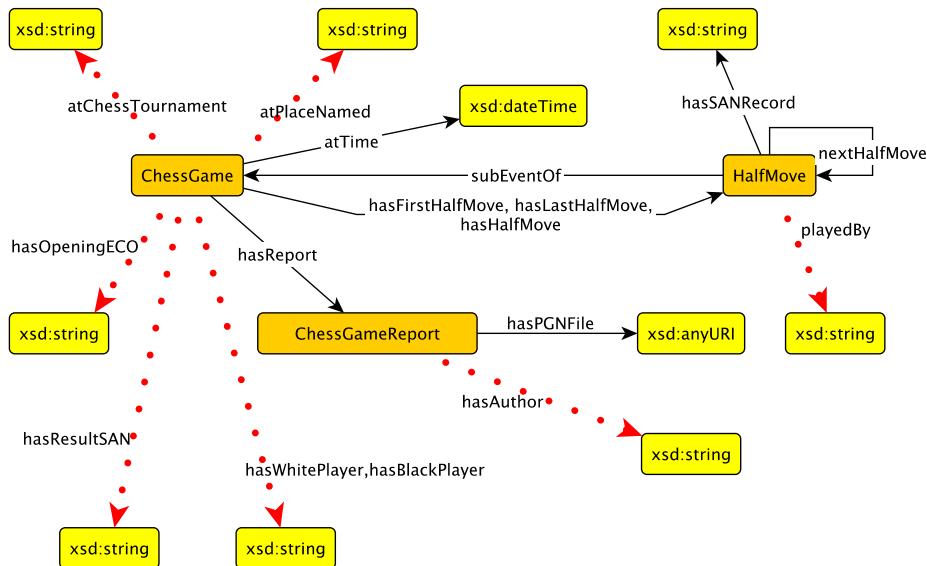


Figure 10.6. Simplified view of the ontology.

2. Data publishers should provide a machine-readable presentation for each URI they maintain in order to enable the URI to be looked up and dereferenced.
3. URIs should be persistent. In the literature, this usually means that they should not contain anything that will likely change such as session tokens or other state information. More generally, however, the persistence of URIs may go beyond that. That is, if a URI should live beyond the lifetime of the underlying infrastructure or the organization who maintains it. In short, once a URI is minted and published online, then it should ideally live forever.
4. URIs should be assumed to be opaque: agents and Web clients accessing a URI should not parse or read into the URI to infer anything about the referenced resource.

Note that URIs in Linked Data are used to identify not just Web documents, but also non-document resources, including real-world objects or even abstract entities. When designing a URI naming scheme, it is thus important to account for the distinction between a thing and the Web document about that thing.

prefix. The latter supposedly indicates that communication to the URI is done with encryption and authentication. Although there is a clear consensus of the need for a secure web for everyone, which motivated projects such as HTTPS Everywhere (<https://www.eff.org/Https-everywhere>) and Let's Encrypt (<https://letsencrypt.org/>), there is a debate as to whether a wholesale move to the HTTPS protocol is the best solution in the long run [5].

For example, consider the website of DaSeLab at Wright State University whose structure may look like this:

- `http://dase.cs.wright.edu/` – the homepage of DaSeLab;
- `http://dase.cs.wright.edu/people/adila-krisnadhi` – the homepage of Adila Krisnadhi in DaSeLab;
- `http://dase.cs.wright.edu/people/pascal-hitzler` – the homepage of Pascal Hitzler in DaSeLab.

Here, all three URIs reside in the same *URI namespace*, which is the DaSeLab namespace given by the URI: `http://dase.cs.wright.edu/`. Moreover, the latter two URIs above reside in a sub-namespace of the DaSeLab namespace given by `http://dase.cs.wright.edu/people/`. Namespaces here are simply a means to ensure the global uniqueness of identifiers. More precisely, within the `http://dase.cs.wright.edu/people/` namespace, the Web resource identified by `http://dase.cs.wright.edu/people/adila-krisnadhi` uniquely corresponds to the homepage of Adila Krisnadhi. Furthermore, on the Web, the URI `http://dase.cs.wright.edu/people/` uniquely correspond to a particular part of the DaSeLab namespace. As a result, one could expect that the URI `http://dase.cs.wright.edu/people/adila-krisnadhi` can be resolved only to a unique resource, which is the homepage of Adila Krisnadhi in DaSeLab, and not something else.

Now, we wish to use URIs also to identify DaSeLab and the two people: Adila Krisnadhi and Pascal Hitzler as *objects*. Since the URIs above have been used to identify the *homepages* of the DaSeLab, and the two people, one should not use them as URIs to identify the three entities as *objects*, i.e., different URIs need to be used. This is to avoid the confusion when using the URIs as part of the statements (concretely, RDF triples) in the linked dataset or ontologies: if we use, for example, `http://dase.cs.wright.edu/people/adila-krisnadhi` as the URI for the person Adila Krisnadhi, then the RDF triple

```
<http://dase.cs.wright.edu/people/adila-krisnadhi> foaf:givenName "Adila".
```

would actually be read as “the homepage of Adila Krisnadhi has a given name Adila”.

10.4.1. Hash URIs and 303 URIs

There are two URI schemes commonly used by Semantic Web applications for identifying non-document resources [27]. They are *hash URIs* and *303 URIs*. Both schemes allow the server to serve both a machine-readable and a human-readable presentations of the same resource, depending on the client’s request.

Hash URIs are URIs that contain a *fragment*, a special part of the URI preceded by a hash symbol (#). When a hash URI is retrieved from the server, HTTP protocol specified that the client application has to strip off the fragment part prior to sending the request to the server. In other words, such an URI, which includes the hash, cannot be retrieved directly, and thus does not necessarily identify a Web document. Therefore, we can use it to identify non-document

resources without raising confusion. For example, we could use the following URIs to identify DaSeLab (the organization), Adila Krisnadhi (the person), and Pascal Hitzler (the person), respectively:

- `http://dase.cs.wright.edu/about#daselab`
- `http://dase.cs.wright.edu/about#adila-krisnadhi`
- `http://dase.cs.wright.edu/about#pascal-hitzler`

When a client requests `http://dase.cs.wright.edu/about#adila-krisnadhi`, it strips off the fragment and only requests `http://dase.cs.wright.edu/about` from the server. Without content negotiation, the server can respond by returning a machine-readable RDF document containing triples describing the three resources above. Meanwhile, with content negotiation, it could send back a machine-readable RDF document or a human-readable HTML document, depending on the client's request. In this case, the client can indicate the preference in the `Accept` header by specifying, e.g., `application/rdf+xml` if the former is preferred, or `text/html` if the latter is preferred instead.

An alternative solution is to employ a special HTTP status code, `303 See Other`, hence the URIs are called 303 URIs. On the surface, there is no apparent distinction on 303 URIs except that they typically contain no fragment part. In this scheme, the URIs may look as follows⁸ where we use the URI namespace `http://dase.cs.wright.edu/id/`:

- `http://dase.cs.wright.edu/id/daselab` – DaSeLab, the organization
- `http://dase.cs.wright.edu/id/adila-krisnadhi` – Adila Krisnadhi, the person
- `http://dase.cs.wright.edu/id/pascal-hitzler` – Pascal Hitzler, the person

The intuition is that according to the Web architecture, it is not appropriate to return a 200 status code⁹ when the above URIs are requested by a client because the URIs actually possess no suitable presentation. Nonetheless, it is always more useful to provide more information about the requested resource, which is also in accordance with Linked Data principles. Here, the server should respond with a 303 status code, which indicates a redirection, and the response also includes `Location` header providing the location of the Web document containing information about the requested resource. Content negotiation can then be used to decide whether to serve a machine-readbale RDF document or a human-readable HTML.

For example, when `http://dase.cs.wright.edu/id/adila-krisnadhi` is requested by a client, the server returns a 303 code and redirects the request to the address given by the `Location` header. Here, we have two slightly different variants. In the first one, the server would redirect to a generic document, say `http://dase.cs.wright.edu/doc/adila-krisnadhi`, and then perform the content negotiation. If an RDF document is requested, it then returns an RDF document, say `http://dase.cs.wright.edu/doc/adila-krisnadhi.rdf`,

⁸Note that there is nothing special with the occurrence of `id` as part of the path in the URIs – one could use `data`, `foo`, or any other string to specify the path component of the URIs.

⁹OK status; meaning that the request has succeeded and the requested resource is returned.

while if an HTML document is requested, then it returns an HTML one, say <http://dase.cs.wright.edu/doc/adila-krisnadhi.html>. In the second variant, the server would perform content negotiation immediately. If the machine-readable RDF content is requested, then the server *redirects* to an RDF document, say <http://dase.cs.wright.edu/data/adila-krisnadhi>. Meanwhile, if HTML content is requested, then the server redirects to an HTML document, say <http://dase.cs.wright.edu/people/adila-krisnadhi>.

10.4.2. URI Naming Scheme

The next aspect that needs to be considered is the URI naming scheme. This aspect is, however, less clear than the URI scheme. Several organizations and communities have provided their own recommendations, conventions, and guidelines, which differ from each other. This is of course not surprising given that in the end, every URI is maintained by a particular party that is also responsible to provide an infrastructure that ensures the URI can be dereferenced.

For example, W3C uses a URI naming scheme of the form, among others, <http://www.w3.org/YYYY/MM/ssss> where YYYY and MM are decimal digits representing the year and month of URI allocation, and ssss is a short string [6].¹⁰ The UK government uses <http://{domain}/id/{concept}/{reference}> as a naming scheme for URIs representing identifiers of entities in the UK government data [9], e.g., <http://education.data.gov.uk/id/school/78> is an identifier of some school in the UK. More examples of the different URI formats as well as their design rules and managements across several government agencies and standardization bodies can be found in a survey by Archer, et al. [1], while a general style guidelines that takes into account multilingual web can be consulted from a paper by Montiel-Ponsoda, et al. [22]. In such design rules and guidelines, one can also usually find recommendations such as preference of short URIs over longer ones, which case policy should be chosen (e.g., all lower case letters, CamelCase, etc.), how to deal with word separator if a URI is formed from several words, when using code value is appropriate, how versioning can be handled, etc. Before we proceed though, we need to choose appropriate namespaces for our identifiers.

If the data as well as ontologies being published are produced and maintained by a particular organization, one could use that organization's URI namespaces for the data and elements of the schema. This is usually a good choice assuming that the organization is not temporary in nature, and there is a clear policy that ensures the persistence of the URIs in case there is change in the underlying infrastructure. Unfortunately, this is in the end impossible to guarantee: organizations may cease to exist, and infrastructure and policies may change. Thus, relying on an organization's URI namespace may prove tricky in the long run.

To help extending the lifetime of URIs beyond the lifetime of the maintaining organizations, the Linked Data community has been using community-supported online services that provide permanent re-direction for Web applications. Such services facilitate data publishers to reserve a URI that is independent of the

¹⁰Quoted directly from the document: “ssss is a short string not causing confusion, alarm, or embarrassment.”

actual address of the server that performs the dereferencing. When a user requests such a URI, the service will redirect the request to the actual location of the information on the Web. If this actual location changes, due to organizational, policy, or infrastructure changes, the URI maintainer can simply modify the redirection in the service to point to the new location. The URI itself is not changed, and hence, does not affect any user applications depending on the URI. Such services have been historically called Persistent Uniform Resource Locators (PURL) services,¹¹ and more recently, there is an ongoing community effort to provide a more unified form of such a service, called w3id,¹² through the W3C Permanent Identifier Community Group.¹³

For our example workflow here, we wish to distinguish two general categories of identifiers. The first category consists of ontology elements, i.e., classes, properties, and named individuals defined by the ontology. The second comprises non-literal resources in the data that are not vocabulary terms from the ontology. For both categories, we take the liberty of claiming <https://w3id.org/rdfchess/> as the top-level namespace.¹⁴ For ontology elements, we assign a sub-namespace given by the ontology names. For non-literal resources that are not a vocabulary term, we reserve <https://w3id.org/rdfchess/id/> sub-namespace. Furthermore, following some of the guidelines from Montiel-Ponsoda, et al. [22], we decide on the following naming scheme.

- We use meaningful local names for ontology elements, and opaque names for data instances. For the latter, we use randomly generated strings as part of the identifiers.
- We employ CamelCase for class and property names from the ontology and lower case alphanumeric letters for named individuals from the ontology and non-literal in the RDF datasets. For the latter, underscore character ('_') is used as the token separator.
- We also employ lower case alphanumeric letters for naming the ontology where the dash character ('-') is used as the token separator if needed.
- We employ hash URIs for the ontology elements and 303 URIs for instances in the data.
- As much as possible, we provide human-readable labels for each resource in the ontology and linked datasets using `rdfs:label` property.
- We ignore versioning issues of the ontology elements to simplify the discussion.

The resulting URI patterns are provided below.

- The pattern <https://w3id.org/rdfchess/id/ssss> is used for non-literal resources in the linked datasets that are not a vocabulary term. Here, `ssss` is a random, unambiguous string. For example, we could use the

¹¹A community site at <http://www.purlz.org/> listed several PURL services, including the oldest one called `purl.org`, which is hosted at OCLC and has been used for more than 15 years by the community.

¹²<https://w3id.org>; unlike most of other PURL services, this one operates in HTTPS-only mode.

¹³<https://www.w3.org/community/perma-id/>

¹⁴Of course, this namespace needs to be *actually* reserved, i.e., by actually performing all the steps specified in <https://w3id.org>

URI <https://w3id.org/rdfchess/id/gam19e02> to represent an instance of the class `ChessGame`; the string `gam19e02` is generated such that within the <https://w3id.org/rdfchess/id/> namespace, it unambiguously refers to this particular instance of `ChessGame`.

- The pattern <https://w3id.org/rdfchess/ontology-name> is used for the name of ontologies. For example, the URI of the complex version of our dual schema could be <https://w3id.org/rdfchess/chessonto>.
- The pattern <https://w3id.org/rdfchess/ontology-name#ClassName> is employed for class names defined in the ontology named `ontology-name`. For example, the `ChessGame` class in the ontology named above could have the URI: <https://w3id.org/rdfchess/chessonto#ChessGame>
- <https://w3id.org/rdfchess/ontology-name#propertyName> is used for property names defined in the ontology named `ontology-name`. For example, <https://w3id.org/rdfchess/chessonto#hasHalfMove> could be used for the `hasHalfMove`.
- https://w3id.org/rdfchess/ontology-name#named_individual is the URI pattern for named individuals defined in the ontology `ontology-name`. E.g., <https://w3id.org/rdfchess/chessonto#grandmaster> could be used to refer to the grandmaster title of chess players. Note that, although named individuals in the ontology are logically treated in the same way as instances of a class in the linked dataset that populates the ontology,¹⁵ they use a different URI pattern. The reason is that named individuals are directly declared in the ontology, usually because they represent some controlled vocabulary in the domain; they are not viewed as data.

10.5. Publishing the Schema

After specifying the URI patterns, the next step is to prepare the OWL file(s) that contain all the axioms in the ontology. Since we are using dual schema, we essentially have two slightly different ontologies corresponding to Fig. 10.1 and 10.6, which can be merged into one according to Figure 10.4. So, our approach is to prepare the following OWL files. These OWL files can be created manually using any ontology editor, such as Protégé¹⁶ or NeOn Toolkit.¹⁷ Alternatively, one can also generate the OWL files programmatically using libraries such as OWL API.¹⁸ For our example workflow, we shall use Protégé to create the OWL files. Please consult the RDF and OWL Primer in the appendix of this book [21] as well as the appropriate documentation of the ontology editor of choice for more information on how to write OWL axioms into the OWL files.

- `chessonto.owl`

This OWL file captures the ontology according to Fig. 10.1, which forms the complex version of the dual schema. The corresponding axioms are given by Fig. 10.2. As ontology URI, we assign <https://w3id.org/rdfchess/chessonto>.

¹⁵In the description logic *lingua*, they are all ABox individuals.

¹⁶<http://protege.stanford.edu/>

¹⁷<http://neon-toolkit.org/>

¹⁸<http://owlcs.github.io/owlapi/>

The URIs for class names, property names, and named individuals in this OWL file are defined according to the patterns in the previous section.

- **chessonto-view.owl**

This OWL file captures the ontology according to Fig. 10.6, expressing a simplified version of the dual schema. As ontology URI, we assign <https://w3id.org/rdfchess/chessonto-view>. Note that since this ontology defines the shortcuts involving classes and properties which are already in `chessonto.owl`, it does not introduce new URIs for those classes and properties, and instead, simply reuses the URIs already defined in `chessonto.owl`. For newly introduced properties such as `hasWhitePlayer`, `hasBlackPlayer`, etc., this ontology declares URIs for them according to the pattern in the previous section. Observe that all of the newly introduced properties in `chessonto-view.owl` are datatype properties. Hence, none of the rules in Fig. 10.5 can be expressed as OWL axioms. Instead, we can write them as DL-Safe SWRL rules [13], which are supported at least by OWL API and Protégé.¹⁹ For example, the first rule in Fig. 10.5 can be expressed as the following SWRL rule where the prefix `chon:` refers to <https://w3id.org/rdfchess/chessonto#> and `chonv:` refers to <https://w3id.org/rdfchess/chessonto-view#>:

```
chon:ChessGame(?x) ~ chon:providesAgentRole(?x,?y)
  ~ chon:WhitePlayerRole(?y) ~ chon:performedBy(?y,?z)
  ~ chon:Agent(?z) -> chonv:hasWhitePlayer(?x,?z)
```

- **chessonto-full.owl**

This OWL file captures the ontology according to Fig. 10.4. We assign <https://w3id.org/rdfchess/chessonto-full> as the ontology URI. This ontology does not define any class, property, or named individual. Instead, it simply imports the previous two OWL files.

The first OWL file is intended for users who wish to populate the Chess ontology before the shortcuts were created. The second is intended for users who prefer to populate the 'shortcut' version. Meanwhile, the third eases users who wish to access the whole dual schema.

After the OWL files are created, we need to publish them on the Web. This publishing step is not strictly needed for generating the linked datasets. Nevertheless, it is necessary if we wish to comply to the Linked Data principles. In particular, we need to ensure that the URIs of the ontologies as well as classes, properties, and individuals in the OWL files to be Web dereferenceable. Since we are using hash URIs, it suffices if the content negotiation module of the Web server²⁰ on which we host the OWL files is correctly configured so that whenever an ontology URI is requested, the correct OWL file is served, e.g., if <https://w3id.org/rdfchess/chessonto> is requested, then the server should serve `chessonto.owl` to the client. In this setting, whenever a class/property/individual name is requested, HTTP protocol would automatically strip off

¹⁹See <https://github.com/protegeproject/swrlapi/wiki/SWRLLanguageFAQ>

²⁰For example, see <http://httpd.apache.org/docs/current/content-negotiation.html> for more information on how to configure content negotiation on the popular Apache web server.

```
[Event "WCh 2013"]
[Site "Chennai IND"]
[Date "2013.11.09"]
[Round "1"]
[White "Carlsen, Magnus"]
[Black "Anand, Viswanathan"]
[Result "1/2-1/2"]
[WhiteTitle "GM"]
[BlackTitle "GM"]
[WhiteElo "2870"]
[BlackElo "2775"]
[ECO "A07"]
[Opening "Reti"]
[Variation "King's Indian attack"]
[WhiteFideId "1503014"]
[BlackFideId "5000017"]
[EventDate "2013.11.09"]

1. Nf3 d5 2. g3 g6 3. Bg2Bg7 4. d4 c6 5. 0-0 Nf6 6. b3 0-0 7. Bb2Bf5 8. c4
Nbd7 9. Nc3 dxc4 10. bxc4 Nb6 11. c5 Nc4 12. Bc1 Nd5 13. Qb3 Na5 14. Qa3 Nc4 15.
Qb3 Na5 16. Qa3 Nc4 1/2-1/2
```

Figure 10.7. Example of PGN file for conversion to RDF triples. The file is obtained from <http://www.pgnmentor.com/events/WorldChamp2013.pgn> and we assume the author of the file is PGN Mentor

the fragment part that represents the class name and simply request the corresponding ontology to the server. If one wishes to have a nicer, human-readable presentation, one could set it up by manually creating an HTML page representing the ontology, or install a wrapper such as LODE²¹ into the content negotiation.

10.6. Publishing the Linked Datasets on the Web

After the vocabulary/schema is ready, we can now populate it by generating the linked datasets. For our Chess example, we generate triples that correspond to the PGN files. Consider Fig. 10.7, which is another example of a PGN file (cf. Fig. 1.1 from Chapter 1). We shall convert data in this PGN file into a set of RDF triples. Since we two slightly different ontologies as part of the dual schema, we can proceed from either way. We shall describe how we can directly populate `chessonto.owl` (i.e., according to Fig. 10.1). Directly populating `chessonto-view.owl` (i.e., according to Fig. 10.6) is analogous. We shall also explain how we can actually populate `chessonto-view.owl` using the RDF triples that populates `chessonto.owl`, and vice versa.

²¹<http://www.essepuntato.it/lode>

10.6.1. Populating A Dual Schema: From the Complex Version to a Simplified Version

Recall that `chessonto.owl` represents the complex version of our dual schema, while `chessonto-view.owl` represents its simplified version. Our first approach to publish a linked dataset is by first populating `chessonto.owl` directly, and afterwards, populating `chessonto-view.owl` by making use of the result of populating `chessonto.owl`. An alternative approach is in the opposite direction, which we shall explain in Section 10.6.2.

10.6.1.1. Populating `chessonto.owl` directly

Guided by the PGN specification²², we can see that the content of the PGN file contains several pieces of information. We go through them one by one below and identify which ontology elements from Fig. 10.1 can be populated. We assume that the base prefix to be `https://w3id.org/rdfchess/id/`, while `xsd` refers to `http://www.w3.org/2001/XMLSchema#`.

- First of all, we create an instance of `ChessGame`, say `:gam19e02`.
- The name of the event in which the chess game was played is “WCh 2013”. So we create an instance for `ChessTournament`, say `:tou23as5`, and assign for `hasName` property the literal value “WCh 2013”²²`xsd:string`. We connect `:gam19e02` to `:tou23as5` via `subEventOf` property.
- The site of the event is the city of Chennai, India, which is also the site of the game. We create an instance representing Chennai for the class `Place`, say `:pla537es8`, and assigns “Chennai”²²`xsd:string` as the value of `hasName`. We then attach it to the instance of `:gam19e02` and `:tou23as5` via `atPlace` property.
- The starting date of the chess game is November 10, 2013. Thus, we attach “2013-11-09T00:00:00”²²`xsd:dateTime`, a literal of type `xsd:dateTime` to `:gam19e02` via `atTime` property. The starting date of the event is also November 9, 2013, so we attach “2013-11-09T00:00:00”²²`xsd:dateTime` to `:tou23as5` via `atTime` property. Note of course that this is a simplification since the event obviously has an end date, though not specified in the PGN file.
- The round of the game in the context of the chess competition is the 1st round. We ignore this information since no classes or property corresponds to it.
- The name of the white player is “Carlsen, Magnus”. Here, we create an instance of `WhitePlayerRole`, say `:rol88y76c`, and attach it to `:gam19e02` via `providesAgentRole` property. We then create an instance of `Agent`, say `:ag422yt6`, attach it to `:rol88y76c` via `performedBy` property, and give “Carlsen, Magnus”²²`xsd:string` as the literal value for `hasName` property. We ignore the title of the white player, which is grandmaster (“GM”).
- The name of the black player is “Anand, Viswanathan”, so we use the literal value “Anand, Viswanathan”²²`xsd:string` for `hasName`.

²²http://www.thechessdrum.net/PGN_Reference.txt

- The result of the game is “1/2-1/2” (i.e., a draw). We create an instance of `ChessGameResult`, say `:res23tu77h`, and then attach the literal value `"1/2-1/2"^^xsd:string` for `encodedAsSAN` property.
- The ELO rating of both the white and black players at the time of the game are 2870 and 2775, respectively. We ignore this information since it does not correspond to any classes or properties.
- The ECO code of the opening of the game is B18. So, we create an instance of `ChessGameResult`, say `:ope662vn2`, and then attach the literal value `"A07"^^xsd:string` for `hasECOCode` property.
- The file contain a numbered sequence of moves where each move consists of two half moves (except possibly the last one) with the white player’s half move preceding the black player’s half move. So, for each pair of half moves, we create an instance of `HalfMove`, attach an appropriate literal value representing the SAN encoding of the half move, and then attach that instance of `HalfMove` to `:gam19e02` using `subEventOf` and `hasHalfMove` (respecting the direction of the property as specified in Fig. 10.1). Instances representing two consecutive half moves should be connected via `nextHalfMove` property. The `hasFirstHalfMove` and `hasLastHalfMove` are used as appropriate to attach the first and last half moves to `:gam19e02`. Finally, for each instance of `HalfMove` we created, we attach a fresh instance of `ActingPlayerRole` via `providesAgentRole` property, and then we connect it to either the instance of `Agent` representing the white chess player (Magnus Carlsen) or the instance of `Agent` representing the black player (Viswanathan Anand). Both of these instances of `Agent` have been created earlier.
- We also create an instance of `ChessGameReport`, add an instance of `AuthorRole` and the corresponding instance of `Agent` representing PGN Mentor, and use <http://www.pgnmentor.com/events/WorldChamp1984.pgn> as the URI of the PGN file.

The above steps result in a set of overall 257 RDF triples given in Fig. 10.8, 10.9, 10.10, and 10.11. As the reader can see, the triples use properties and classes from `chessonto.owl`.

Naturally, writing them all by hand would be rather tedious. The mapping between tags in the PGN file and the vocabulary terms in the ontology also has to be set manually. Fortunately, there are already a number of programming libraries and APIs that can help us in generating and manipulating RDF triples, e.g., Apache Jena API (Java),²³ RDF4J (Java; formerly known as Sesame),²⁴ Redland (C),²⁵ rdflib (Python),²⁶ etc.

10.6.1.2. Storing RDF triples

The triples just generated can be stored as a file dump. The most recent version of RDF, i.e., RDF 1.1, specified several standard serialization formats:²⁷

²³<https://jena.apache.org/>

²⁴<http://rdf4j.org/>

²⁵<http://librdf.org/>

²⁶<https://github.com/RDFLib/rdflib/>

²⁷The older RDF 1.0 only has RDF/XML as the standard serialization format.

```

@prefix : <https://w3id.org/rdfchess/id/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix chon: <https://w3id.org/rdfchess/chessonto#> .

:gam19e02 rdf:type chon:ChessGame ;
    chon:subEventOf :tou23as5 ;
    chon:atPlace :pla537es8 ;
    chon:atTime "2013-11-09T00:00:00"^^xsd:dateTime ;
    chon:providesAgentRole :rol88y76c, :rol92z01m ;
    chon:hasResult :res23tu77h ;
    chon:hasOpening :ope662vn2 .

:tou23as5 rdf:type chon:ChessTournament ;
    chon:hasName "WCh 2013"^^xsd:string ;
    chon:atPlace :pla537es8 ;
    chon:atTime "2013-11-09T00:00:00"^^xsd:dateTime .

:pla537es8 rdf:type chon:Place ;
    chon:hasName "Chennai"^^xsd:string .

:rol88y76c rdf:type chon:WhitePlayerRole ; chon:performedBy :ag422yt6 .

:ag422yt6 rdf:type chon:Agent ;
    chon:hasName "Carlsen, Magnus"^^xsd:string .

:rol92z01m rdf:type chon:BlackPlayerRole ; chon:performedBy :ag79yy12 .

:ag79yy12 rdf:type chon:Agent ;
    chon:hasName "Anand, Viswanathan"^^xsd:string .

:res23tu77h rdf:type chon:ChessGameResult ;
    chon:encodedAsSAN "1/2-1/2"^^xsd:string .

:ope662vn2 rdf:type chon:ChessOpening ;
    chon:hasECOCode "A07"^^xsd:string .

:cgr448uy6 rdf:type chon:ChessGameReport ;
    chon:providesAgentRole :rol08jj2a ;
    chon:hasPGNFile <http://www.pgnmentor.com/events/WorldChamp2013.pgn> .

:rol08jj2a rdf:type chon:AuthorRole ;
    chon:performedBy :ag66bn89 .

:ag66bn89 rdf:type chon:Agent ;
    chon:hasName "PGN Mentor"^^xsd:string

```

Figure 10.8. RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 — continued in Fig. 10.9

```

:gam19e02 chon:hasFirstHalfMove :hmgam19e021a ;
    chon:hasLastHalfMove :hmgam19e0216b ;
    chon:hasHalfMove
        :hmgam19e021a, :hmgam19e021b, :hmgam19e022a, :hmgam19e022b,
        :hmgam19e023a, :hmgam19e023b, :hmgam19e024a, :hmgam19e024b,
        :hmgam19e025a, :hmgam19e025b, :hmgam19e026a, :hmgam19e026b,
        :hmgam19e027a, :hmgam19e027b, :hmgam19e028a, :hmgam19e028b,
        :hmgam19e029a, :hmgam19e029b, :hmgam19e0210a, :hmgam19e0210b,
        :hmgam19e0211a, :hmgam19e0211b, :hmgam19e0212a, :hmgam19e0212b,
        :hmgam19e0213a, :hmgam19e0213b, :hmgam19e0214a, :hmgam19e0214b,
        :hmgam19e0215a, :hmgam19e0215b, :hmgam19e0216a, :hmgam19e0216b .

:hmgam19e021a rdf:type chon:HalfMove ; chon:hasSANRecord "Nf3"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9021a ; chon:nextHalfMove :hmgam19e021b .
:rolhg19e9021a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e021b rdf:type chon:HalfMove ; chon:hasSANRecord "d5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9021b ; chon:nextHalfMove :hmgam19e022a .
:rolhg19e9021a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e022a rdf:type chon:HalfMove ; chon:hasSANRecord "g3"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9022a ; chon:nextHalfMove :hmgam19e022b .
:rolhg19e9022a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e022b rdf:type chon:HalfMove ; chon:hasSANRecord "g6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9022b ; chon:nextHalfMove :hmgam19e023a .
:rolhg19e9022b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e023a rdf:type chon:HalfMove ; chon:hasSANRecord "Bg2"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9023a ; chon:nextHalfMove :hmgam19e023b .
:rolhg19e9023a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e023b rdf:type chon:HalfMove ; chon:hasSANRecord "Bg7"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9023b ; chon:nextHalfMove :hmgam19e024a .
:rolhg19e9023b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e024a rdf:type chon:HalfMove ; chon:hasSANRecord "d4"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9024a ; chon:nextHalfMove :hmgam19e024b .
:rolhg19e9024a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e024b rdf:type chon:HalfMove ; chon:hasSANRecord "c6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9024b ; chon:nextHalfMove :hmgam19e025a .
:rolhg19e9024b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e025a rdf:type chon:HalfMove ; chon:hasSANRecord "O-O"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9025a ; chon:nextHalfMove :hmgam19e025b .
:rolhg19e9025a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

```

Figure 10.9. (continued from Fig. 10.8) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 – continued in Fig. 10.10

```

:hmgam19e025b rdf:type chon:HalfMove ; chon:hasSANRecord "Nf6"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9025b ; chon:nextHalfMove :hmgam19e026a .
:rolhg19e9025b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e026a rdf:type chon:HalfMove ; chon:hasSANRecord "b3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9026a ; chon:nextHalfMove :hmgam19e026b .
:rolhg19e9026a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e026b rdf:type chon:HalfMove ; chon:hasSANRecord "0-0"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9026b ; chon:nextHalfMove :hmgam19e027a .
:rolhg19e9026b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e027a rdf:type chon:HalfMove ; chon:hasSANRecord "Bb2"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9027a ; chon:nextHalfMove :hmgam19e027b .
:rolhg19e9027a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e027b rdf:type chon:HalfMove ; chon:hasSANRecord "Bf5"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9027b ; chon:nextHalfMove :hmgam19e028a .
:rolhg19e9027b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e028a rdf:type chon:HalfMove ; chon:hasSANRecord "c4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9028a ; chon:nextHalfMove :hmgam19e028b .
:rolhg19e9028a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e028b rdf:type chon:HalfMove ; chon:hasSANRecord "Nb7"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9028b ; chon:nextHalfMove :hmgam19e029a .
:rolhg19e9028b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e029a rdf:type chon:HalfMove ; chon:hasSANRecord "Nc3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9029a ; chon:nextHalfMove :hmgam19e029b .
:rolhg19e9029a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e029b rdf:type chon:HalfMove ; chon:hasSANRecord "dxc4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e9029b ; chon:nextHalfMove :hmgam19e0210a .
:rolhg19e9029b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0210a rdf:type chon:HalfMove ; chon:hasSANRecord "bxc4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90210a ; chon:nextHalfMove :hmgam19e0210b .
:rolhg19e90210a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0210b rdf:type chon:HalfMove ; chon:hasSANRecord "Nb6"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90210b ; chon:nextHalfMove :hmgam19e0211a .
:rolhg19e90210b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0211a rdf:type chon:HalfMove ; chon:hasSANRecord "c5"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90211a ; chon:nextHalfMove :hmgam19e0211b .
:rolhg19e90211a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

```

Figure 10.10. (continued from Fig. 10.9) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 – continued to Fig. 10.11

```

:hmgam19e0211b rdf:type chon:HalfMove ; chon:hasSANRecord "Nc4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90211b ; chon:nextHalfMove :hmgam19e0212a .
:rolhg19e90211b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0212a rdf:type chon:HalfMove ; chon:hasSANRecord "Bc1"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90212a ; chon:nextHalfMove :hmgam19e0212b .
:rolhg19e90212a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0212b rdf:type chon:HalfMove ; chon:hasSANRecord "Nd5"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90212b ; chon:nextHalfMove :hmgam19e0213a .
:rolhg19e90212b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0213a rdf:type chon:HalfMove ; chon:hasSANRecord "Qb3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90213a ; chon:nextHalfMove :hmgam19e0213b .
:rolhg19e90213a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0213b rdf:type chon:HalfMove ; chon:hasSANRecord "Na5"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90213b ; chon:nextHalfMove :hmgam19e0214a .
:rolhg19e90213b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0214a rdf:type chon:HalfMove ; chon:hasSANRecord "Qa3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90214a ; chon:nextHalfMove :hmgam19e0214b .
:rolhg19e90214a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0214b rdf:type chon:HalfMove ; chon:hasSANRecord "Nc4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90214b ; chon:nextHalfMove :hmgam19e0215a .
:rolhg19e90214b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0215a rdf:type chon:HalfMove ; chon:hasSANRecord "Qb3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90215a ; chon:nextHalfMove :hmgam19e0215b .
:rolhg19e90215a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0215b rdf:type chon:HalfMove ; chon:hasSANRecord "Na5"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90215b ; chon:nextHalfMove :hmgam19e0216a .
:rolhg19e90215b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0216a rdf:type chon:HalfMove ; chon:hasSANRecord "Qa3"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90216a ; chon:nextHalfMove :hmgam19e0216b .
:rolhg19e90216a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0216b rdf:type chon:HalfMove ; chon:hasSANRecord "Nc4"^^xsd:string ;
  chon:providesAgentRole :rolhg19e90216b .
:rolhg19e90216b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

```

Figure 10.11. (continued from Fig. 10.10) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1

RDF/XML, N-triples, Turtle (which we used in this chapter), RDFa, JSON-LD, TriG, and N-Quads. Most APIs for manipulating RDF would one to read and write those serializations formats. This would be sufficient if we just want to do some offline processing, do not need to pose custom queries to the data, and the number of triples is not too large that it is feasible to load them all in memory. In many other situations, we have to store the RDF triples in a triple store. One can choose any triple store product, both the open source as well as proprietary ones. Examples of triple stores as well as storage solutions that support storing RDF data include OpenLink Virtuoso,²⁸ TDB from Apache Jena,²⁹ RDF4J (formerly known as Sesame), Stardog,³⁰ Redland, GraphDB,³¹ BrightStarDB,³² Strabon,³³ Blazegraph³⁴, Dydra,³⁵ Mulgara,³⁶ CubicWeb,³⁷ AllegroGraph,³⁸ MarkLogic,³⁹ etc. These storage solutions support data retrieval using some query languages. Most of them support SPARQL and the others support other variants of graph querying language. Some additionally allows one to set up a SPARQL endpoint that is directly accessible from the Web typically via some kind of API, such as REST API.

10.6.1.3. Populating `chessonto-view.owl` using the populated `chessonto.owl`

Now that we have generated RDF triples to populate `chessonto.owl`, we can use them to populate `chessonto-view.owl`. The idea is to make use of the shortcuts we defined in Fig. 10.5. Note that `chessonto-view.owl` already contains a set of DL-safe SWRL rules expressing the shortcuts. Thus, one approach is to employ a SWRL or Datalog reasoner, e.g., Drools⁴⁰ or RDFox⁴¹ to materialize the conclusions of each of those rules over the combination of the RDF triples and `chessonto-view.owl` as the underlying knowledge base, i.e., declaring all URIs in the RDF triples that are neither a class nor a property to be an OWL named individual. The materialization will result in a set of generated triples, which can then be added back to the original set of RDF triples.

Another approach is to either use any RDF API to search for the triples matching the body of the rules and then generate triples corresponding to the head of the rules. For example, the first rule in that figure is:

$$\begin{aligned} \text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{WhitePlayerRole}(y) \wedge \text{performedBy}(y, z) \\ \wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasWhitePlayer}(x, s) \end{aligned}$$

²⁸<http://virtuoso.openlinksw.com/>

²⁹<https://jena.apache.org/documentation/tdb/>

³⁰<http://stardog.com/>

³¹<http://graphdb.ontotext.com/>

³²<http://brightstardb.com/>

³³<http://www.strabon.di.uoa.gr/>

³⁴<https://www.blazegraph.com/>

³⁵<http://dydra.com/>

³⁶<http://www.mulgara.org/>

³⁷<https://www.cubicweb.org/>

³⁸<http://franz.com/agraph/allegrograph/>

³⁹<http://www.marklogic.com/>

⁴⁰<http://www.drools.org/>

⁴¹<http://www.cs.ox.ac.uk/isg/tools/RDFox/>

```
?x rdf:type chon:ChessGame ;
  chon:providesAgentRole ?y .
?y rdf:type chon:WhitePlayerRole ;
  chon:performedBy ?z .
?z rdf:type chon:Agent ;
  chon:hasName ?s .
```

Figure 10.12. Triple patterns for the body of the first rule from Fig. 10.5. The prefix chon: refers to <<https://w3id.org/rdfchess/chessonto#>>

```
CONSTRUCT {
  ?x chonv:hasWhitePlayer ?s
} WHERE {
  ?x rdf:type chon:ChessGame ;
    chon:providesAgentRole ?y .
  ?y rdf:type chon:WhitePlayerRole ;
    chon:performedBy ?z .
  ?z rdf:type chon:Agent ;
    chon:hasName ?s .
}
```

Figure 10.13. SPARQL CONSTRUCT query for the first rule from Fig. 10.5. The prefix chon: and chonv: refer to <<https://w3id.org/rdfchess/chessonto#>> and <<https://w3id.org/rdfchess/chessonto-view#>>, respectively.

The body of this rules can be generally read as a set of triple patterns. That is, it corresponds to the set of triple patterns given in Fig. 10.12.

One could also run SPARQL CONSTRUCT queries, which can be created for each rule, on the RDF triples, assuming that they are stored in a triple store. For example, the above rule can be expressed in the query given in Fig. 10.13. Running this query will yield a set of triples that can be added back to the original set of RDF triples, which now also populates chessonto-view.owl.

10.6.2. Alternative approach: Populating Dual Schema from a Simplified Version to the Complex Version

This approach may be appealing for some data providers because we start by populating chessonto-view.owl, which represents a simpler version of the dual schema. To populate chessonto-view.owl directly, the steps are essentially the same as the ones we took to directly populate chessonto.owl as described in Section 10.6.1.1. The only notable difference is that we do not need to generate a URI for WhitePlayerRole, providesAgentRole, Agent, etc. Be mindful that some of the classes and properties in chessonto-view.owl have the same URIs as those in chessonto.owl.

The more interesting part of this approach is the subsequent step where we can populate chessonto.owl based on the populated chessonto-view.owl. This

```

CONSTRUCT {
  ?x rdf:type chon:ChessGame ;
    chon:providesAgentRole
      [ rdf:type chon:WhitePlayerRole ;
        chon:performedBy
          [ rdf:type chon:Agent ;
            chon:hasName ?s ]
      ] .
} WHERE {
  ?x chonv:hasWhitePlayer ?s .
}

```

Figure 10.14. A SPARQL CONSTRUCT query expressing the reverse direction of the first rule from Fig. 10.5 where blank nodes are generated. The prefix `chon:` and `chonv:` refer to [<https://w3id.org/rdfchess/chessonto#>](https://w3id.org/rdfchess/chessonto#) and [<https://w3id.org/rdfchess/chessonto-view#>](https://w3id.org/rdfchess/chessonto-view#), respectively.

time, however, we cannot use the SWRL rules in `chessonto-view.owl`, at least directly, because the process corresponds to the backward direction of the rules. That is, the triples we have correspond to the head of the rules and we have to generate new triples that correspond to the body of the rules. Therefore, SWRL reasoners cannot help us here. Fortunately, we can still approach it via RDF APIs and SPARQL CONSTRUCT queries. For example, based on the rule:

$$\begin{aligned}
\text{ChessGame}(x) \wedge \text{pAR}(x, y) \wedge \text{WhitePlayerRole}(y) \wedge \text{performedBy}(y, z) \\
\wedge \text{Agent}(z) \wedge \text{hasName}(z, s) \rightarrow \text{hasWhitePlayer}(x, s)
\end{aligned}$$

we can write a SPARQL CONSTRUCT query in Fig. 10.14 that intuitively runs in the opposite direction of the rule. Running this on the triples populates a part of `chessonto-view.owl`. Such a query can generally be created for each rule in Fig.. 10.5.

Observe that we generate blank nodes for `WhitePlayerRole` and `Agent`. If we wish to generate actual URIs, we need to hard-code parts of the URI namespaces for the instances of `WhitePlayerRole` and `Agent` and employ a number of SPARQL built-in functions. Generally, this also requires the help of some naming convention to ensure the unambiguity of the generated URIs within the same namespace. An example of the SPARQL query is given in Fig. 10.15.

After we generate the triples, we still have to do some post-processing, because according to `chessonto.owl` as also depicted in Fig. 10.1, the agent that acts as the white player (resp. black player) of a chess game should be the same as the agent that acts as the acting player of the half move of a white player (resp. black player) in the chess game.

For example, using the example of the chess game discussed in this chapter (as given by Fig. 10.7) and the SPARQL CONSTRUCT queries that generate blank nodes (the query in Fig. 10.14 and another similar query that is based on last rule in Fig. 10.5), we would have generated the triples in Fig. 10.16 where `:gm123` is the URI representing the instance of `ChessGame` that corresponds to

```

CONSTRUCT {
  ?x rdf:type chon:ChessGame ;
    chon:providesAgentRole ?roleuri .
  ?roleuri rdf:type chon:WhitePlayerRole ;
    chon:performedBy ?agenturi .
  ?agenturi rdf:type chon:Agent ;
    chon:hasName ?s .
} WHERE {
  ?x chonv:hasWhitePlayer ?s .
  BIND (STRUUID() AS ?roleid)
  BIND (STRUUID() AS ?agentid)
  BIND (URI(CONCAT("https://w3id.org/rdfchess/id/", "rol", ?roleid))
    AS ?roleuri)
  BIND (URI(CONCAT("https://w3id.org/rdfchess/id/", "ag", ?agentid))
    AS ?agenturi)
}

```

Figure 10.15. A SPARQL CONSTRUCT query expressing the reverse direction of the first rule from Fig. 10.5 that generates actual URIs, instead of blank nodes. The prefix `chon:` and `chonv:` refer to `<https://w3id.org/rdfchess/chessonto#>` and `<https://w3id.org/rdfchess/chessonto-view#>`, respectively. We use, as a part of naming convention, the string “`rol`” and “`ag`” as the initial part of the name of instances of `AgentRole` and `Agent`, respectively.

the discussed chess game and `:move1` is the URI representing the instance of `HalfMove` that corresponds to the first half move of the chess game, while `_:x`, `_:x1`, `_:y`, `_:y1` are all blank nodes. There, the blank nodes `_:x` and `_:y` must correspond to the same instance of `Agent`. Thus, we need to replace `_:x` with `_:y` or vice versa.

An analogous post-processing also needs to be done if we employ the SPARQL CONSTRUCT queries that generate actual URIs.

10.6.3. Redundancy in the Data and Linking with Other RDF Data

We now obtained a linked dataset that populate the dual schema. One problem that may still happen is redundancy inside the data, especially if the RDF triples are programmatically generated from the PGN files. This redundancy is mainly caused by the fact that two different PGN files may contain the same information that is expressed using different string values. One obvious example is the name of chess players. PGN Specification does not specify a fixed format for a chess player’s name. One PGN file may contain “Carlsen, Magnus”, while another may contain “Magnus Carlsen”, and both refer to the same chess player Magnus Carlsen. If we can find two URIs that should correspond to the same entity, we can assert a link (i.e., as a triple) between these two entities using `owl:sameAs` predicate or weaker predicates such as the ones from the SKOS vocabulary.⁴²

The above problem also extends to a more general problems of linking with

⁴²<http://www.w3.org/2004/02/skos/core>

```

:gm123 rdf:type chon:ChessGame ;
    chon:providesAgentRole _:x1 .
_:x1 rdf:type chon:WhitePlayerRole ;
    chon:performedBy _:x .
_:x rdf:type chon:Agent ;
    chon:hasName "Carlsen, Magnus"^^xsd:string .
:gm123 chon:hasHalfMove :move1 .
:move1 chon:providesAgentRole _:y1 .
_:y1 rdf:type chon:ActingPlayerRole ;
    chon:performedBy _:y .
_:y rdf:type chon:Agent ;
    chon:hasName "Carlsen, Magnus"^^xsd:string .

```

Figure 10.16. Example RDF triples that need post-processing: `_:x` and `_:y` must correspond to the same instance of `Agent`.

other linked datasets in the Linked Open Data Cloud.⁴³ The latter is important to make our linked dataset truly in the five-star category [7]. As an example, we shall augment our dataset with links to some other linked dataset in the LOD Cloud. The simplest one to pick is DBpedia dataset, which is based on Wikipedia infoboxes. If we inspect our dataset, we can intuitively guess that a few pieces of information correspond to real world entities that may appear in Wikipedia. In particular, we may be able to find information about chess players, the chess tournaments, and location of the chess tournaments in DBpedia since those information are likely to be available in Wikipedia. With regards to the RDF triples we generated in this chapter, we find the following URIs from DBpedia datasets:

- http://dbpedia.org/resource/Magnus_Carlsen representing the chess player Magnus Carlsen;
- http://dbpedia.org/resource/Viswanathan_Anand representing the chess player Viswanathan Anand;
- http://dbpedia.org/resource/World_Chess_Championship_2013 representing the World Chess Championship 2013;
- <http://dbpedia.org/resource/Chennai> representing the city of Chennai, India.

Hence, we can enrich our dataset by adding triples given in Fig. 10.17 to the ones we already obtained in Fig. 10.8, 10.9, 10.10, and 10.11.

One could easily see that finding links to other linked datasets as exemplified above can also be understood as detecting whether two URIs refer to the same entity. In our example above, we are quite fortunate since we are able to find the corresponding URIs in DBpedia manually. If we want to do this to every set of triples we generated from the PGN files, then we need to do it programmatically, and the general computational problem corresponds to the so-called *coreference resolution*, which is beyond the scope of this chapter. Nevertheless, this section

⁴³<http://lod-cloud.net/>

```
:ag422yt6 owl:sameAs <http://dbpedia.org/resource/Magnus_Carlsen> .
:ag79yy12 owl:sameAs <http://dbpedia.org/resource/Viswanathan_Anand> .
:tou23as5 owl:sameAs <http://dbpedia.org/resource/World_Chess_Championship_2013>
:pla537es8 owl:sameAs <http://dbpedia.org/resource/Chennai>
```

Figure 10.17. Links to DBpedia data using `owl:sameAs` where the prefix `owl:` refers to `<http://www.w3.org/2002/07/owl#>`. If `owl:sameAs` is felt too strong, one can use other linking vocabulary such as the one from SKOS.

demonstrates the advantage of Linked Data for combining and integrating our chess dataset with other existing datasets.

10.6.4. Serving the Data on the Web

We can publish the triples we just generated as a downloadable file (RDF dumps) and putting it somewhere accessible by everyone on the Web. We assume that the ontologies as well as the vocabulary terms are all Web-dereferenceable. So, we just need to ensure that other URIs appearing in any of the RDF triple, which is of the form `https://w3id.org/rdfchess/id/ssss`, are Web-dereferenceable. This can be achieved by setting up an appropriate content negotiation so that whenever such URIs are requested, the RDF dump is returned. This is certainly a very crude way to serve the data, though arguably already satisfies the Linked Data principles.

A nicer way, albeit putting more burden on the data provider, is to set up a Linked Data infrastructure that host the RDF triples. The backend of such an infrastructure is typically a triple store that we already touched upon in one of the earlier section, although other types of database are also possible. The triple store is accessible from one or more endpoints that accept queries from users, usually expressed in SPARQL. On top of a SPARQL endpoint, one can then set up a front-end that, with appropriate Web server configuration, can handle content negotiation nicely. In addition, such a front-end allows one to either perform a follow-your-nose browsing of the data or formulate custom queries directly on the data. More fancy front-ends also provides visualization functionalities. One can certainly develop such a front-end in-house, but there are examples of front-end products that one can also use or adapt, e.g., Linked Media Framework,⁴⁴ D2RQ,⁴⁵ Pubby,⁴⁶ OpenLink Virtuoso's front-end,⁴⁷ PublishMyData,⁴⁸ Linked Data Fragments,⁴⁹ Exhibit,⁵⁰ etc.

⁴⁴<https://bitbucket.org/srfgkmt/lmf/>

⁴⁵<http://d2rq.org/>; intended to provide access to relational databases as virtual RDF graphs

⁴⁶<http://wifo5-03.informatik.uni-mannheim.de/pubby/>

⁴⁷<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/>

VirtDeployingLinkedDataGuide_UsingVirtuoso

⁴⁸<http://www.swirrl.com/>

⁴⁹<http://linkeddatafragments.org/>

⁵⁰<http://www.simile-widgets.org/exhibit3/>

10.7. Conclusion

This chapter presented a rather high-level description on the steps that one needs to take to publish Linked Data that is based on Ontology Design Patterns. Some technical pointers also have been provided. Obviously, we do not cover more specific technical details, which can normally be obtained from the documentation of the Linked Data infrastructure products that we referred to earlier. A rather prominent issue that we do not address concerns the possible redundancy of information particularly if we wish to generate one large linked dataset that covers not just one particular PGN file, but rather, a large collection of ones, e.g., the whole collection of PGN files from PGN Mentor. The way we generated RDF triples as discussed in this chapter assumes that for each PGN file, we generate a new URI for the chess tournament, the chess players (as instances of `Agent`), location of the tournament (as instance of `Place`), the author of the game description (as instance of `Agent`), and the chess opening that has its own ECO code. It is rather easy to see that those entities may occur across different PGN files, e.g., there may be more than one PGN files in which Magnus Carlsen appears as either the white or black player.

To address this redundancy, we can modify the procedure of generating RDF triples so that we do not look at one PGN file as an isolated data source, but look at one collection of PGN files as a whole. When generating the triples, we then need to reuse URIs for a particular entity (players, tournaments, places, etc.) if we encountered one that we have seen in the collection before. Alternatively, we can still consider each PGN file by itself, and after we finish processing all PGN files, we perform a post-processing step where we search for URIs that represent the exact same entity, and then add an `owl:sameAs` triple for each pair of such URIs. Both alternatives, however, boil down to coreference resolution problem, which generally may not be trivial, and thus, is beyond the scope of this chapter.

Acknowledgements. This work was partially supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*, by the EC under the International Research Staff Exchange Scheme (IRSES) of the EU Marie Curie Actions, project *SemData – Semantic Data Management*, and by the Spanish Ministry of Economy and Competitiveness (project TIN2013-46238-C4-2-R).

Bibliography

- [1] P. Archer, N. Loutas, S. Goedertier, and S. Kourtidis. Study on persistent URIs with identification of best practices and recommendations on the topic for the Member States and the European Commission, June 24, 2013. Available at <http://philarcher.org/diary/2013/uripersistence/>.
- [2] O. Bartlett. Linked Data: Connecting together the BBC’s online content. BBC Internet Blog, February 13, 2013. Available at <http://www.bbc.co.uk/blogs/internet/entries/af6b613e-6935-3165-93ca-9319e1887858>.

- [3] O. Bartlett. Olympic Data services and the interactive video player. BBC Internet Blog, July 31, 2012. Available at <http://www.bbc.co.uk/blogs/internet/entries/92c2ec84-6ec0-33f2-9dc7-0f915d28ff52>.
- [4] W. Beek, L. Rietveld, H. R. Bazoobandi, J. Wielemaker, and S. Schlobach. LOD laundromat: A uniform way of publishing other people's dirty data. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, editors, *The Semantic Web – ISWC 2014 – 13th International Semantic Web Conference, Riva del Garda, Italy, October 19–23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2014.
- [5] T. Berners-Lee. Web security - "HTTPS Everywhere" harmful. Available at <https://www.w3.org/DesignIssues/Security-NotTheS.html>, 15 February 2012.
- [6] T. Berners-Lee. *URIs for W3C Namespaces*. W3C Technical Report Publication Policies, 25 April 2006. Available at <https://www.w3.org/2005/07/13-nsuri>.
- [7] T. Berners-Lee. Linked data. Available at <https://www.w3.org/DesignIssues/LinkedData.html>, 27 July 2006.
- [8] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2015.
- [9] P. Davidson. Designing URI sets for the UK Public Sector: A report from the Public Sector Information Domain of the CTO Council's cross-government enterprise architecture. Interim paper version 1.0, Chief Technology Officer Council, October 2009. Available at https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/60975/designing-URI-sets-uk-public-sector.pdf; Last accessed: May 12, 2016.
- [10] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.
- [11] S. Harris and A. Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. Available at <https://www.w3.org/TR/sparql11-query/>.
- [12] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.
- [13] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, 21 May 2004. Available from <http://www.w3.org/Submission/SWRL/>.
- [14] B. Hyland, G. Atemezing, and B. Villazón-Terrazas, editors. *Best Practices for Publishing Linked Data*. W3C Working Group Note, 09 January 2014. Available at <https://www.w3.org/TR/ld-bp/>.
- [15] K. Janowicz, P. Hitzler, B. Adams, D. Kolas, and C. Vardeman. Five stars of linked data vocabulary use. *Semantic Web*, 5(3):173–176, 2014.

- [16] A. Krisnadhi. *Ontology Pattern-Based Data Integration*. PhD thesis, Wright State University, 2015.
- [17] A. Krisnadhi and P. Hitzler. Modeling with Ontology Design Patterns: Chess games as a worked example. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [18] A. Krisnadhi, F. Maier, and P. Hitzler. OWL and Rules. In A. Polleres et al., editors, *Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 382–415. Springer, Heidelberg, 2011.
- [19] A. A. Krisnadhi, P. Hitzler, and K. Janowicz. On the capabilities and limitations of OWL regarding typecasting and ontology design pattern views. In V. A. M. Tamma, M. Dragoni, R. Gonçalves, and A. Lawrynowicz, editors, *Ontology Engineering - 12th International Experiences and Directions Workshop on OWL, OWLED 2015, co-located with ISWC 2015, Bethlehem, PA, USA, October 9-10, 2015, Revised Selected Papers*, volume 9557 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2015.
- [20] M. Krötzsch, F. Maier, A. A. Krisnadhi, and P. Hitzler. A better uncle for OWL: Nominal schemas for integrating rules and ontologies. In S. Sadagopan, K. Ramamritham, A. Kumar, M. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th International World Wide Web Conference, WWW2011, Hyderabad, India, March/April 2011*, pages 645–654. ACM, New York, 2011.
- [21] F. Maier. A primer on RDF and OWL. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [22] E. Montiel-Ponsoda, D. Vila-Suero, B. Villazón-Terrazas, G. Dunsire, E. E. Rodriguez, and A. Gómez-Pérez. Style guidelines for naming and labeling ontologies in the multilingual web. In T. Baker, D. I. Hillmann, and A. Isaac, editors, *Proceedings of the 2011 International Conference on Dublin Core and Metadata Applications, DC 2011, The Hague, The Netherlands, September 21-23, 2011*, pages 105–115. Dublin Core Metadata Initiative, 2011.
- [23] J. Rayfield. BBC World Cup 2010 dynamic semantic publishing. BBC Internet Blog, July 12, 2010. http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html.
- [24] L. Rietveld, W. Beek, R. Hoekstra, and S. Schlobach. Meta-data for a lot of LOD. *Semantic Web*, 2016. Conditionally accepted for publication.
- [25] V. Rodríguez-Doncel, A. Gómez-Pérez, and N. Mihindukulasooriya. Rights declaration in linked data. In O. Hartig, J. Sequeda, A. Hogan, and T. Matsutsuka, editors, *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*, volume 1034 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
- [26] V. Rodríguez-Doncel, A. A. Krisnadhi, P. Hitzler, M. Cheatham, N. Karima, and R. Amini. Pattern-based linked data publication: The linked chess

- dataset case. In O. Hartig, J. Sequeda, and A. Hogan, editors, *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015.*, volume 1426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [27] L. Sauermann and R. Cyganiak, editors. *Cool URIs for the Semantic Web*. W3C Interest Group Note, 03 December 2008. Available at <https://www.w3.org/TR/cooluris/>.
 - [28] M. Schmachtenberg, C. Bizer, and H. Paulheim. State of the LOD cloud 2014. <http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/>, 30 August 2014.
 - [29] E. Winter. Copyright on chess games. <http://www.chesshistory.com/winter/extra/copyright.html>, 1987.
 - [30] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.

Chapter 11

Modeling Ontology Design Patterns with Domain Experts – A View From the Trenches

Krzysztof Janowicz, University of California, Santa Barbara, USA

11.1. Introduction and Motivation

Ontology engineering methods, frameworks, and tools have been studied for more than 20 years and have been widely applied in academia, industry, and at government agencies. While there are considerable differences, most established methods share a common core, namely a knowledge acquisition activity that spans over phases such as (requirements) specification, conceptualization, and formalization with the goal of effectively eliciting knowledge from domain experts to implement an ontology [4, 10, 15, 19–22]. Despite substantial progress, knowledge acquisition remains a key bottleneck and has been extensively studied in work on expert systems since the 1980s [14]. Some of the underlying problems such as scaling can be addressed by a combination of classical top-down engineering with bottom-up, data-driven techniques [5]. Today, this is increasingly important for arriving at a deeper axiomatization of Linked Data to improve interoperability and query federation.

However, one can also look at the problem of acquiring knowledge from domain experts from an entirely different angle and argue that the challenge lies in reaching agreement between domain experts [12] together with the misconception that there has to be a common understanding of terminology within a particular domain. While there should be a broad and domain overarching agreement on units of measure, the same cannot be said about concepts such as *environment*, *resilience*, *forest*, *city*, and so forth. This is not only true across domains but even within research fields and communities [10]. This misconception may have its roots in the successful standardization of protocols, formats, and languages which may have caused ontology engineers and domain experts to believe that one could and should also *standardize meaning* on the Web.

The meaning of terms, however, depends on different viewpoints, cultures, underlying themes, legislation, state of knowledge, level of detail, current context, and many other aspects that jointly lead to the vast heterogeneity of data published on the Web today. To give an intuitive and commonly used example, a street can be conceptualized as a *connection* between two places from the view point of transportation science or as a *separation* that cuts a habitat into pieces from the view point of ecology. This heterogeneity is not a problem that needs to be *resolved* but a resource that enables us to address challenges that cannot be fully understood by considering one of these aspects in isolation. Nonetheless, some common core is needed to retrieve, reuse, and integrate data across different sources. In other words, communication and thus interoperability break down [17] if there is no overlap between the underlying conceptualizations of the shared (physical) space. As *specifications* of these conceptualizations, ontologies provide us with the means to approximate commonalities and differences between datasets, communities, domains, and so forth. Simply put, the purpose of ontologies is to make the intended meaning *explicit*, not to *agree* on what terms mean.

Following this argumentation, the key challenge for ontology engineering becomes striking a balance between fostering interoperability without restricting semantic heterogeneity, i.e., without enforcing a common understanding. One approach is to defer the introduction of concepts that are heavy on *ontological commitments*, e.g., *vulnerability*, and first focus on the development, reuse, and combination of common building blocks [10]. The resulting ontologies reflect the conceptualizations of the data providers and still ensure a minimum fallback level for interoperability via the usage of these common building blocks. Content ontology design patterns [6] are a promising candidate for such an approach and also act as a middle ground between engineering local ontologies from scratch and trying to reuse existing ontologies despite their differing ontological commitments [2, 16].

In this work, we describe our experience in modeling such content patterns with domain experts from a variety of different domains and fields including geography, oceanography, geology, industrial ecology, transportation science, architecture, the digital humanities, and so forth. The presented work reflects on the lessons learned and challenges of running more than a dozen vocabulary camps (VoCamps) and similar events over the past six years. While we will focus on multi-day vocabulary camps here, the discussion can be generalized to other ontology engineering events that bring domain experts and ontology engineers together.

In the following, we will address the *why, what, and how* of pattern modeling with domain experts. First, we will briefly revisit the *perceived* value proposition of ontologies and Semantic Web technologies for domain experts to better understand their expectations and reasons for participating in the development of ontologies. We will then discuss how VoCamps are structured and why we believe they are a successful approach to pattern engineering. Finally, using the Semantic Trajectory pattern [8], we will outline modeling choices by example and highlight how they foster reusability and flexibility of patterns.

11.2. The Value Proposition of Ontologies

In the following we will revisit the value proposition of ontologies from the view point of individual domain experts (here researchers), in contrast to large-scale data providers, government agencies, industry, and so forth, to better understand *why* they become interested in ontologies and participate in vocabulary camps. We will structure the value proposition into three different stages: publishing and retrieving data, interacting with data, and reusing and integrating data. Note that in contrast to previous work [11], we focus on the *perceived* value proposition.

11.2.1. Publishing and Retrieving Data

The added value of ontologies starts with publishing own data. There are at least three different motives to semantically annotate these data by means of ontologies. First, researchers hope to improve the discoverability of their scientific results beyond mere keyword search. This aspect goes hand in hand with the hope to be able to better and faster retrieve useful data from others. As a consequence, many domain experts approach ontology engineering from the perspective of building richer taxonomies. Second, an increasing number of funding organizations, scientific journals, and universities require data publication and managing strategies. Semantic Web technologies are perceived as a promising choice for doing so. The thirds and final reason is less obvious despite having the most important long-term effects. By semantically annotating their data, scientists improve the reproducibility of their results, e.g., by adding provenance records, and reduce the risk of (accidental) misinterpretation and therefore wrong usage of their own data [18]. Based on our observations, many domain experts interested in ontology engineering have worked with databases and conceptual modeling environments before and thus may falsely expect that ontologies are primarily used for integrity constraints.

11.2.2. Interacting with Data

Knowledge exploration, e.g., by follow-your-nose browsing or faceted search, are common ways to interact with Linked Data and in many cases more attractive and intuitive to domain experts than querying endpoints using SPARQL. A common hope associated with Linked Data is that the available data hubs will store different information about common entities and thus enable a more holistic view. What is often overlooked, however, is the fact that these data sources will likely contain overlapping information. To give an example, there will be multiple population counts for a populated place and multiple geographic coordinates for its centroid. Given the decentralized nature of the Web and current research focus, a majority of exciting frameworks do not yet consider Linked Data fusion/conflation [1]. Consequently, the perceived value proposition of Semantic Web technologies differs from the research focus. From an ontology modeling perspective, one could argue that a stronger axiomatic foundation will ease Linked Data fusion, e.g., by identifying functional properties. For now, Linked Data users have to handle contradicting data themselves.

11.2.3. Reusing and Integrating Data

Discovering and interacting with data on the Web is often a means to an end with the reuse and integration of the data being the final goal. A very common misconception among domain experts is the belief that the creation of Linked Data and here especially the use of owl:SameAs will magically enable query federation. The realization that even a densely linked global graph of data does not necessarily enable queries over multiple of the involved data hubs often comes as a shock [9]. Understandably, the first reaction is to call for common top-level and domain ontologies to ensure that the same classes and properties are used across these data sources. It takes a deeper understanding along the lines of argumentation made in the introduction section and a basic understanding of ontology *alignment* to realize why domain ontologies do not exist for the vast majority of research fields and why those that exist are facing substantial challenges. A second misconception exists with respect to the formal semantics of knowledge representation languages such as OWL. As discussed above, many domain experts and data providers approach ontologies from an integrity constraints or object-oriented design perspective and do not fully understand the consequences of an inferential semantics and the Open World Assumption. The most common example are (global) domain and range restrictions and the belief that they would constrain the usage of properties. This is one of the reasons why vocabulary camps make use of *guarded* domains and range restrictions instead [13].

11.3. Vocabulary Camps for Pattern Engineering

In this section we will introduce VoCamps as a means to develop content patterns and outline *how* they are set up. Most of what will be said can be generalized to related forms of synchronous knowledge acquisition.

11.3.1. General Setup and Scope

The events organized over the past years were scheduled for 2-3 days. Here we will outline a 2-day event. VoCamps are *unconferences*, i.e., there is no registration fee, no formal presentations (aside of selected invited talks), no proceedings, and so forth. Instead VoCamps focus on bringing domain experts and ontology engineers together to address real modeling issues. (Geo)VoCamps typically draw between 20 and 40 attendees. While the composition varies substantially, arriving at a balanced combination of ontology engineers and domain experts was not a problem so far. Ideally there would be one experienced ontology engineer per 2-3 domain experts. At least in case of GeoVoCamps, a team of 6-8 regular participants ensures continuity and brings in the required expertise.

VoCamps typically start with domain experts that bring their modeling problems to the event. From these, 2-3 are selected and addressed during the VoCamp. The key success to a productive event is to have relatively short but intense break-out sessions with frequent reports back to uncover similarities and differences with the other groups/patterns and to identify potential issues and improvements. The

Table 11.1. A representative agenda for a VoCamp.

	1st day	2nd day
Morning	<ul style="list-style-type: none"> • Introduction • Report from previous VoCamps • A selected pattern as example • Invited talks 	<ul style="list-style-type: none"> • Brief recap • Breakout groups work on patterns • Reports from the groups • Breakout groups (<i>implementation</i>)
Afternoon	<ul style="list-style-type: none"> • Decide on breakout groups • Breakout groups work on patterns • Reports from the groups 	<ul style="list-style-type: none"> • Reports from the groups • Breakout groups (<i>examples/data</i>) • Reports from the groups • Brief documentation

goal of a VoCamp is typically the conceptualization and draft axiomatization of a pattern as well as a brief documentation together with examples. In the weeks following a VoCamp, the results are improved and polished and finally published as an (OWL) ontology together with a paper documenting and motivating the work. Follow-up VoCamps then take up on this work to align their own patterns, learn from the modeling and implementation approaches taken before, or refine certain aspects of the pattern. Experience from the last years show that at least 2 of 3 patterns started at a VoCamp are eventually published.

It is too early to determine the long-term impact of the developed patterns and the VoCamp model.¹ However, some of them, e.g., the trajectory or agent-role patterns, have been used in various settings. At this stage, we believe that bringing domain experts and ontology engineers together and spreading an understanding for the possibilities and limitations of Semantic Web technologies is the core outcome. The uptake of ontologies (and patterns more specifically) in large cyber-infrastructure such as NSF's Earthcube as well as a growing interest among domain experts from multiple disciplines provides positive signals.

11.3.2. VoCamp Agenda and Workflow

Table 11.1 outlines a representative agenda of a VoCamp. The morning of the first day is used to introduce the unconference-style to new participants, to report on previous VoCamps and their results thereby ensuring continuity, as well as a brief presentation outlining the design and implementation of a pattern as example. These three items set the stage and scope for the event. This is typically followed by one or two invited short talks of about 20 minutes that introduce an interesting domain or research field or provide a novel perspective on patterns and ontology engineering. Finally, domain experts get a chance to pitch their modeling projects and ideas before the lunch break.

The afternoon of the first day usually begins with selecting 2-4 potential topics/patterns and then splitting into breakout groups. Ideally, each group consists of 6-10 members with at least one experienced ontology engineer, domain expert, data provider, and dedicated scribe (or moderator). During the first breakout session which takes 2-3 hours, each group decides on the scope of their work, i.e.,

¹See also <http://dase.cs.wright.edu/blog/geovocamps-taking-stock>.

where to start and stop modeling a certain topic or domain problem. This phase is also characterized by discussions on the used domain terminology and is the most intense part of the VoCamp as no common ground has been established so far. Explaining that multiple views can be reconciled and that different modeling choices can be transferred into each other often helps to prevent *turf wars* between domain experts. That said, different and contradicting views points brought in by domain experts are the most fruitful aspect of this scoping phase as they will later ensure the reusability and extensibility of the developed pattern. The first afternoon ends with a reporting and feedback session. These sessions are of fundamental importance as allowing members from other groups to actively comment on and contribute to other patterns helps to establish a constructive atmosphere and ensures *buy-in* for the final patterns. Finally, the participants also discuss reoccurring modeling choices and strategies which will further ease the later alignment of the patterns.

The second morning and afternoon sessions follow the same template of intense work within breakout groups and frequent reports. The most important difference, however, is that a draft implementation of the patterns (typically in OWL) has to be developed and presented by the end of the second morning. The afternoon breakout session is then used to populate the pattern with real data and to document the work to a degree where it can be finished in an asynchronous style as the participants come from different institutions and countries.

There is no specific ontology engineering method that is used across all groups and VoCamps. Nonetheless, there are a few common aspects that have emerged as successful strategies. One of them is the formulation of competence questions [7] during the first breakout session and the use of tools to develop concept maps on a shared screen. Working with real data, populating the draft patterns, and writing SPARQL queries to retrieve the data are other commonly used strategies. Finally, some of the core VoCamp members rotate between groups to keep the meeting productive and to ensure that no single domain expert or ontology engineer dominates the discussion.

11.4. Pattern Design Decisions by Example

A discussion of what distinguishes a pattern from a small ontology and how to best engineer reusable patterns is out of scope here. Nonetheless, it is worth looking at a specific example of a pattern designed at a VoCamp to understand why certain choices have been made, how they enable a broad and domain spanning usage of the pattern, and *what* kinds of problems are typically addressed. We will use the Semantic Trajectory pattern as example [8]. More specifically, we will argue how the pattern fulfills the criteria for patterns informally defined as quality and reusability proxies at VoCamps, namely that patterns should:

- Cover a wide range of domains or application areas.
- Be extensible to provide additional details.
- Supports multiple granularities.
- Provide an axiomatization beyond mere surface semantics.
- Have various *hooks* to well-known ontologies / patterns.

- Be self-contained to a degree where they can be used on their own.

It is interesting to note that the second and last point require striking the right balance between developing a pattern that is generic enough to act as a building block for an application ontology but self-contained (and specific) to a degree where a meaningful use, e.g., the creation of Linked Data, does not require additional ontologies. The deeper axiomatization of the pattern is left to a later chapter in this volume.

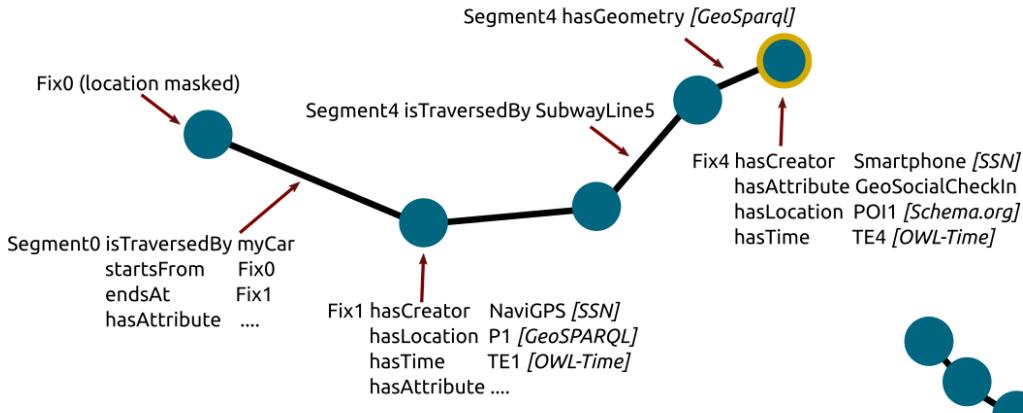
Trajectories play a key role in many areas, be it for wildlife tracking, studies of human movement, traffic analysis, scene modeling, cruises of oceanographic research vessels, health and exposure research, to name but a few. Simply put, a trajectory is the path that a moving object takes through space as a function of time. While such definition also includes the trajectory of a projectile, the pattern is designed for cases that benefit from a semantic annotation of the moving object, the places visited, the mode of transportation, the sensor used to determine the objects location at a given time, and so forth, thus forming a *semantic* trajectory.

To do so, the pattern introduces classes such as *fix*, *segment*, *moving object*, *position*, *source*, and *attribute*, together with relations between them. Fixes, for instance, determine the position of an object at a certain time and are observed by some source. Two successive fixes of the same trajectory are connected by a segment, i.e., a linear interpolation of the path taken. The segments themselves are traversed by a moving object.

Interestingly, as depicted in figure 11.1, this minimal pattern already supports a wide range of use cases. For instance, one could model the path taken from UCSB to the Los Angeles airport as a single segment connecting the start fix Santa Barbara with the end fix Los Angeles. In such case, the fixes are not just arbitrary measurements taken by the used positioning technology, e.g., a GPS-based navigation system, based on the device's sampling rate but meaningful *places*. In this case, the segment represents U.S. Highway 101 and the moving object (if given) could be a car or bus. Such modeling, of course, does not imply that the highway is indeed a linear feature nor that only two fixes were taken. Instead, it reflects the needs, here the resolution, of an application or use case. A typical example would be studying origin-destination trips. On the other extreme, one can approximate the exact path taken by a moving object by increasing the number of fixes and thus shortening the linear segments up to the level supported by the used positioning technology. For example, a modern smartphone can take GPS fixes once every second with an accuracy of about 5 meters. Here a typical use case for the pattern would be recreational hiking. Finally, the pattern can also cover the middle ground. In the first example all fixes were places, while they were mere positioning artifacts in the second case. In many trip planning applications or wildlife monitoring it is important to roughly approximate the path taken and also to identify certain Points Of Interest (POI). Intuitively a fix taken at a fuel station or watering place carries additional meaning. Summing up, the Semantic Trajectory pattern supports multiple granularities and can be used for a wide range of use cases. It is also self-contained in the sense that the described examples can be fully modeled with the pattern alone.

The pattern also provides multiple opportunities for integration with other well established patterns and ontologies. For instance, the pattern specifies that

Human travel trajectory



Abstraction



Discretization

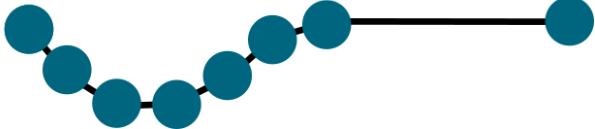


Figure 11.1. Flexible modeling via the Semantic Trajectory pattern.

fixes are created by some *source* but does not provide any further axiomatization for this source. Consequently, the source could be a GPS sensor, a human observer, or even a written itinerary from a historical expedition. If the source is indeed a sensor, it can be further specified, e.g., to describe the frequency (*ssn:Frequency*) using the W3C-XG Semantic Sensor Network ontology (SSN) [3]. Similarly, the pattern does not define place types as this would be out of scope and limit reusability. Instead, one can state that a certain fix was taken at a position within a recognized place of a given type, e.g., using a subclass of DBpedia's *place* class. The pattern also provides a more general way to add domain and application-specific information by allowing every fix and segment to have additional attributes, e.g., the type of street traveled. Summing up, the pattern provides hooks for other patterns and ontologies thereby acting as a true building block that can be easily extended and combined.

Let us finally consider an example that brings the aforementioned points together. Figure 11.1 shows a part of a human trajectory. It starts at a fix that does not provide any further positioning details, e.g., to mask the home location of a user. The first segment of the trajectory is traveled by a car. Fix1 was taken by the car's navigation system at a specific position that can be represented as a point-feature in Well-Known-Text (WKT) serialization and by using OWL-Time. Segment4 was traveled via a specific subway line. So far, none of the segments had an associated geometry. However, this is possible (conform with GeoSPARQL), e.g., by using a LineString. Finally, the last fix was created by a user's smartphone and the position is not given in terms of geographic coordinates but as so-called geo-social check-in to a Point Of Interest, e.g., a bar, using a social network application such as Foursquare/Swarm. We have since used the Semantic Trajectory pattern to model oceanographic cruises, wildlife tracking, and so forth to substantiate the reusability and extensibility claims. In cases, of cruises, for instance, POI are stops at ports and the moving objects are restricted to research vessels.

11.5. Summary

In this chapter we briefly motivated the need for an ontology design pattern-driven approach to ontology engineering by pointing out that Semantic Web technologies have been often misunderstood as tools to reach or enforce a common agreement, i.e., view on the world, while their true nature is in making meaning and thus differences in the interpretation of terms in different communities explicit. We revisited the *perceived* value proposition of ontologies for domain experts to understand their motivation for joining VoCamps and for using Semantic Web technologies. We pointed out that not all of these hopes can be addressed by the current state-of-the-art or because they contradict with essential design choices made at early stages of the Semantic Web. Next, we discussed how VoCamps are structured and the methods they use to develop patterns. Finally, we have presented a pattern developed at a VoCamp to highlight selected design decisions by example.

Bibliography

- [1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data—the story so far. In *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.
- [2] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2015.
- [3] M. Compton, P. Barnaghi, L. Bermudez, R. GarcÃ¡a-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25 – 32, 2012.
- [4] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. Methontology: from ontological art towards ontological engineering, 1997.
- [5] A. Gangemi. A comparison of knowledge extraction tools for the semantic web. In *The Semantic Web: Semantics and Big Data*, pages 351–366. Springer, 2013.
- [6] A. Gangemi and V. Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.
- [7] M. Grüninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Benchmarking – Theory and Practice*, pages 22–31. Springer, 1995.
- [8] Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitzler, M. Dean, and D. Kolas. A geo-ontology design pattern for semantic trajectories. In *Spatial Information Theory*, pages 438–456. Springer, 2013.
- [9] P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, and A. P. Sheth. Linked data is merely more data. In *AAAI Spring Symposium: linked data meets artificial intelligence*, volume 11, 2010.
- [10] K. Janowicz. Observation-driven geo-ontology engineering. *Transactions in GIS*, 16(3):351–374, 2012.
- [11] K. Janowicz and P. Hitzler. Key ingredients for your next semantics elevator talk. In *Advances in Conceptual Modeling*, pages 213–220. Springer Berlin Heidelberg, 2012.
- [12] K. Janowicz, P. Maué, M. Wilkes, S. Schade, F. Scherer, M. Braun, S. Dupke, and W. Kuhn. Similarity as a quality indicator in ontology engineering. In *Proceedings of the 2008 Conference on Formal Ontology in Information Systems: Proceedings of the Fifth International Conference (FOIS 2008)*, pages 92–105, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.
- [13] A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. Finin, P. Ji, M. Jones, N. Karima, K. Lehnert, A. Mickle, T. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The geolink modular oceanography ontology. In M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srivivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayanan, and S. Staab,

- editors, *The Semantic Web - ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 301–309. Springer International Publishing, 2015.
- [14] D. B. Lenat, M. Prakash, and M. Shepherd. Cyc: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI magazine*, 6(4):65, 1985.
 - [15] H. S. Pinto, S. Staab, and C. Tempich. DILIGENT: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, volume 110, page 393, 2004.
 - [16] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. eXtreme design with content ontology design patterns. In *Proc. Workshop on Ontology Patterns, Washington, DC, USA*. Citeseer, 2009.
 - [17] S. Scheider and W. Kuhn. How to talk to each other via computers: Semantic interoperability as conceptual imitation. In *Applications of Conceptual Spaces*, pages 97–122. Springer, 2015.
 - [18] S. Scheider and M. Tomko. Knowing whether spatio-temporal analysis procedures are applicable to datasets. In *9th International Conference on Formal Ontology in Information Systems (FOIS 2016)*. IOS Press, 2016.
 - [19] P. Spyrs, Y. Tang, and R. Meersman. An ontology engineering methodology for dogma. *Applied Ontology*, 3(1-2):13–39, 2008.
 - [20] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1):161–197, 1998.
 - [21] M. C. Suárez-Figueroa, A. Gomez-Perez, and M. Fernandez-Lopez. The NeOn methodology for ontology engineering. In *Ontology engineering in a networked world*, pages 9–34. Springer, 2012.
 - [22] M. Uschold, M. Gruninger, et al. Ontologies: Principles, methods and applications. *Knowledge engineering review*, 11(2):93–136, 1996.

This page intentionally left blank

Chapter 12

Using ODPs for Ontology Transformation

Vojtěch Svátek, Department of Information and Knowledge Engineering
University of Economics, Prague

Ondřej Zamazal, Department of Information and Knowledge Engineering
University of Economics, Prague

Marek Dudáš, Department of Information and Knowledge Engineering
University of Economics, Prague

12.1. Motivation for and Types of Ontology Transformation

A semantic web ontology, as machine-readable encoding of a shared conceptualization of some (usually, real-world) state of affairs, may not always remain stable and unchangeable. One reason for modifying it may be a change in the state of affairs it reflects. Another one may be the requirements of the computational environment (consisting of software applications as well as data structures) in which it is to be processed.

The former kind of change has, in most cases, the character of evolution; we thus speak of *ontology evolution* [19], which typically proceeds in smaller steps that gradually make the first version of the ontology deprecated in favor of new ones. Since ontology evolution is driven by specific content-level requirements, it is tough to abstract higher-level design patterns from it and is thus of lesser interest for the ontology pattern research.

In this chapter we rather deal with ontology modification triggered by requirements of the computational environment (in broader sense, also including human readability concerns etc.), which we denote as *ontology transformation*. There, in contrast, the changes are typically applied in bulk, throughout the whole ontology, possibly introducing new or totally eliminating existing language constructs, or even switching to another representation formalism. The original version of the ontology does not necessarily lose its value, since it may still faithfully reflect the true conceptualization. It is due to the bulk application of transformation and to the language-level nature of the change operations that the potential of using *patterns* is obvious.

Given the extremely vast landscape of what is called ‘ontology’ in today’s computing disciplines, we limit our account to use cases and methods that involve a specific, yet widespread, technological platform: the ontological language OWL,¹ the first version of which has been standardized by the W3C more than a decade ago. OWL is interoperable and largely overlapping with an even more widely used semantic web representation language, RDF [32]. Given the huge ecosystem of applications and tools for these languages, and the public pressure on using RDF wherever on the globe data are to be published in open format (RDF being the 4- and 5-star representation for open data in the rating proposed by Tim Berners-Lee and constantly spreading [13]), we furthermore always consider the *target* format of the transformation to be OWL.²

A broad categorization of the transformation use cases then may look as follows:

- ‘Internal’ transformation *within* OWL, by replacing certain OWL constructs by others, often with impact on the overall expressiveness. For instance, a data property can be replaced with an analogous object property, or an object property can be transformed to a class (a notorious example is the change of the ‘marriedTo’ property to class ‘Marriage’ and two auxiliary properties linking the two spouses). We label this type of transformation as OWL2OWL.
- Transformation of ‘non-ontological’ representation to OWL (NOR2OWL). By ‘non-ontological’ representation (NOR, where ‘R’ may stand for either ‘representation’, or for ‘resource’ as in prior work [31]) we mean shallow models somehow reflecting a shared conceptualization but lacking the grounding in formal logic. These representations, be it thesauri, business taxonomies, government-sphere codelists, or other, largely rely on natural-language descriptions of concepts.
- Transformation of some *other*, typically, less constrained and less operational, *ontological representation* to OWL. Since this other representation is likely to reflect the real-world state of affairs more closely or less ambiguously than the OWL representation resulting from its transformation, we might see it as a kind of *ontological background model* (OBM) behind the ‘foreground’ OWL representation used by applications. Consequently, we nick this transformation species as OBM2OWL. An example is the transformation of an entity explicitly typed as ‘role’, ‘anti-rigid’ or similar in the OBM (e.g., ‘Student’, ‘Employee’ or ‘Food’) into an OWL class – which may be then, in contrast to ‘rigid’ classes, assigned and unassigned to individuals during their lifecycle.

The chapter covers all three mentioned types. However, most space is devoted to OWL2OWL transformation, for several reasons. First, the authors have most extensive experience with it. Second, it is not substantially covered in prior literature. And third, and possibly most important, thanks to the fact that both the input and output of transformation is a single, well-defined (albeit rich) lan-

¹<https://www.w3.org/TR/owl2-overview/>

²This for example eliminates an ultimately interesting problem of generating controlled English statements from OWL ontologies.

guage, the space of underlying transformation patterns tends to be more compact. In contrast, while NOR2OWL transformation is widely applied, and some kind of transformation pattern technology is often relevant for it, we only cover it marginally in the chapter, since the number of different ‘non-ontological’ representations is excessive and each requires a specific (in its ‘source’ part) pattern language. This is somewhat analogous for OBM2OWL transformation.

The following sections are devoted, in turn, to a very brief overview of NOR2OWL transformation, slightly more extended ones for OBM2OWL and OWL2OWL transformation, and, eventually, to a particular OWL2OWL transformation project, PatOMat, explained in more depth. We then wrap up with a brief conclusions section.

12.2. NOR2OWL Transformation Patterns

The problem of transforming non-ontological resources (NOR) to OWL has been thoroughly addressed by Villazón-Terrazas and Gómez-Perez [31] recently. We thus only point out the ‘pattern’ aspect of the discussed approaches, and refer the reader to the cited work. The patterns for transforming non-ontological resources to OWL are characterized there as *re-engineering patterns*, under the assumption that the resulting OWL ontology will make explicit the ontological intentions that are only implicit in the non-ontological model. The two types of NOR considered are *classification scheme*, which is a taxonomic grouping of concepts only relying on a single type of relationship (the taxonomic one), and *thesaurus* or *lexicon* (the two treated equivalently), in which the concepts (or, rather, terms or words) may be connected via different relationships, including hierarchical (hyperonymy, meronymy, etc.) and non-hierarchical (such as synonymy, antonymy or general ‘relatedness’) ones. The cited work presents sixteen general re-engineering patterns, classified by two main facets:

- The *source* structure, which can be,
 - for a *classification scheme*, one of: *path enumeration* (path from root stored in a single field of the node record), *adjacency list* (parent listed in the child record), *snowflake* (hierarchy levels in separate tables) or *flattened* (hierarchy levels in separate columns)
 - for a *thesaurus* or *lexicon*: either a *relation-based* (allowing to specify an arbitrary relation) or a *record-based* (with dedicated fields for pre-defined relations only) structure
- The *target* structure, which can be,
 - an ontological Tbox alone, or
 - a Tbox together with Abox.

In the ‘Tbox’ approach, the ‘terms’ from NOR are generally converted to OWL classes. The challenge in converting a NOR to an OWL ontology is then in correctly identifying the type of relationship to be constructed in the target ontology. For example, the ‘broader than’ (BT) and ‘narrower than’ (NT) relationships in a thesaurus are typically transformed to *owl:subClassOf*, the ‘related term’ (RT)

relationship is transformed to an object property³ labelled as *relatedClass*, and the ‘used for’ (UF) relationship makes one term an additional value for *rdfs:label* of the class generated from the other, non-preferred term.

12.3. OBM2OWL Transformation Patterns

Since the advent of ontological modeling based on reasoner-friendly logical languages (OWL and its description-logic predecessors, as well as languages based on other grounding, such as OCML [16]), most knowledge engineers have been aware of the modeling constraints incurred by the language expressiveness limits, which have been, in turn, dictated by tractability concerns. A common practice in neat ontological projects had been, in a period, to initially draft ontological axioms in first-order logic before tweaking them to more constrained (taxonomic-oriented) representations. There has not however been substantial tool support for this transformation operation. With the widespread of OWL, the developers of (in some sense) more powerful or ‘deeper’ ontological modeling languages sought a certain level of interoperability with OWL in terms of ‘downgrading’ the developed models to this comparably ‘lightweight’ language. We will briefly review some of these approaches, putting specific emphasis on the possibility to use declarative patterns in this transformation.

12.3.1. OBO to OWL transformation

OBOnTO is an ontology format developed within the *Open Biomedical Ontologies* initiative, which stemmed from the Gene Ontology effort. There is a conversion service⁴ for transforming OBO ontologies into OWL. While some logical relationships from OBO, such as intersection/union/disjoint classes, domain/range of relation or symmetric/transitive relations can be straightforwardly translated to the corresponding OWL constructs, many other are only captured via annotation properties. The transformation patterns are informally described in a mapping table, but there does not seem to be any declarative operational representation of them.

12.3.2. OntoUML to OWL transformation

OntoUML is a version of UML tailored for ontology-driven conceptual modeling, developed by G. Guizzardi [8]. The design of OntoUML models is supported by the OLED editor [7], which allows to generate OWL code. There are three generation options, which can be viewed as transformation pattern families. The first, straightforward one, maps OntoUML classes, associations and attributes to OWL classes, object properties and data properties. The second, originally based on the work by Zamborlini et al., [35] specifically focuses on representing temporal information in OWL. Finally, the third [1] does not generate OWL proper but

³Actually, in DL terms, a meta-property connecting individuals meta-modeling the classes under the same identifier (a modeling technique known as punning).

⁴http://www.bioontology.org/wiki/index.php?title=OboInOwl:Main_Page

expresses the additional OntoUML features (not expressible in OWL) using the SWRL rule language⁵.

12.3.3. PURO to OWL transformation

Compared to the previous two kinds of background modeling, PURO [27] is considerably simpler and shallower. The acronym reflects the two basic distinctions: *Particular* vs. *Universal* and *Relationship* vs. *Object*. There is also a third possibility added to the relationship/object distinction – *valuation*, which is an assignment of a quantitative value. The combination of these distinctions determines the six basic PURO terms:⁶

- \mathcal{B} -object (particular object),
- \mathcal{B} -type ('universal object', i.e. type of objects; or even type of types),
- \mathcal{B} -relationship (particular relationship),
- \mathcal{B} -relation (type of relationships),
- \mathcal{B} -valuation (particular assertion of quantitative value), and
- \mathcal{B} -attribute (type of valuations).

A PURO OBM is composed of instances of these six primitives and of the relationships *subTypeOf* and *instanceOf*. The modeling is ‘by example’: while the OBM also contains universals, they are glued with (real-world, artificial, or just placeholder) particulars.

The PURO metamodel is analogous to OWL, which reduces its added value as ‘richer’ modeling language in the background, but on the other hand eases its understanding by semantic web ontology designers. It can be roughly said that \mathcal{B} -object corresponds to OWL individual, \mathcal{B} -type to class and \mathcal{B} -relation to property. There are however purposeful differences that allow PURO to abstract from encoding choices that are necessary in OWL; this makes PURO amenable as background modeling language for OWL despite its much weaker inventory of ‘Tbox’ primitives. Namely, PURO allows for higher-order types (i.e. types having types as their instances) and, as it is not limited by the data model of RDF, does not have the arity of relationships limited to two. As consequence, for example, relationships that require reification in OWL can be expressed directly in PURO. Similarly, the ‘classes as property values’ problem [18] not only disappears⁷ as a \mathcal{B} -type can naturally participate in a \mathcal{B} -relation, but PURO also to some degree allows to express, at background level, inherently different motivations of making a class a property value [26]. For example, a book individual that is linked to the ‘Lion’ class by the property ‘hasSubject’ may talk about concrete, enumerated lions, or about unspecified lions (as ‘topic’), or about the zoological taxon of ‘lion’ as discussed by scholars. Obviously, PURO also overarches minor syntactic varieties arising in OWL models, such as the use of data properties instead of object properties for linking to notions (e.g., countries) represented with string codes rather than IRI identifiers (which is common, e.g., in markup vocabularies).

⁵<https://www.w3.org/Submission/SWRL/>

⁶The ‘ \mathcal{B} ’ stands for ‘background’.

⁷This effect can be achieved in OWL itself by allowing OWL Full or syntactically via so-called punning.

Automatic generation of OWL code from a PURO model yields basic declarations of classes (possibly with instances – but normally dropping the ‘placeholder’ or even ‘sample’ ones), subclass axioms and (object and data) properties with domain and range. Complex Tbox axioms, in turn, are assumed to be manually added to the generated OWL representation. While not being a way to author an ontology in bulk, PURO background modeling can serve as ‘spotlight’ on the different modeling options, some of which might be omitted from consideration otherwise.

Different OBM2OWL patterns [3] can be explicitly chosen to guide the transformation (i.e. OWL generation) process. Using these patterns, for example, a simple PURO fragment consisting of a book related to a ‘published in’ property to a city in which it was published, can be translated to at least four different OWL fragments⁸ differing in their style:

- *data property*: d:b1 o:publishedIn "Prague".
- *object property*: d:b1 o:publishedIn d:Prague.
- *reified relationship*: d:_rr1 a o:publishedIn; o:book d:b1;
o:place data:Prague.
- *dedicated class instantiation*: d:b1 a o:BookPublishedInPrague.

Note that, in this case, none of the target styles is completely contrived but actually corresponds to the style used by an actual bibliographic or generic (e.g., *schema.org* or *DBpedia*) ontology.

Support for PURO modeling and subsequent transformation to OWL is provided by a recently developed prototype tool suite, PURO Modeler and OBOWL-Morph [4]. Technically, the transformation is carried out using SPARQL UPDATE queries, whose structure closely reflects the transformation patterns.

12.4. OWL2OWL Transformation patterns

If we analyze the possible reasons for OWL2OWL ontology transformation, we can identify, among other, the following ones:

- Improving the current ontology’s *tractability* by reasoners, i.e. replacing troublesome constructs by more palatable ones or removing them overall.
- Improving the current ontology’s *understandability* for a human.
- *Repairing* the current ontology by removing an instance of *anti-pattern* (as recurring undesired structure usually leading to some kind of incoherence).
- Introducing a *best-practice artifact*, such as a design pattern or core ontology, into the current ontology, which typically asks for a structural change of it.
- *Aligning* the current ontology to another ontology, in order to let them share their instances, allow for federated querying, or merging them outright.

⁸In the OWL fragments, the ‘d’ prefix always refers to a fictional dataset and ‘o’ to a fictional ontology.

Another, partly correlated and more general dimension could obviously be that of the nature of the update operations used within the transformation: whether additions, removals or both are applied, and which of them is primordial. In this respect we can distinguish between:

- *Pure addition*: the purpose is to add new statements in bulk; this kind of transformation is common and usually least challenging.
- *Removal-centric transformation*: the purpose is to remove some undesired statements, be they instances of anti-patterns or structures that merely cause troubles to reasoners. Additions then however may be required to compensate for the lost inferences.
- *Style transformation*: statements are introduced that are not ‘new’ (in contrast to ‘pure addition’ transformation) but represent a different way of expressing the content of some of the existing statements. The original statements may or may not be removed, however, in the former case the removal is not the primordial operation (in contrast to ‘removal-centric’ transformation).

A long-term project that addressed most of the space of these two dimensions, but with special focus on style transformation, is *PatOMat* [34], to which we devote a whole section below. However, we first survey other tools and approaches that also comply with the notion of (at least to some degree, pattern-based) OWL2OWL transformation.

There are several programming interfaces enabling changes in an ontology, which are typically applied in ‘pure addition’ mode. *OPPL* (Ontology Pre-Processing Language) is a macro language, based on Manchester OWL syntax,⁹ for manipulating ontologies written in OWL. OPPL was introduced in [11] and applied in [5]. Its initial purpose was to provide a declarative language to enrich lean ontologies with automatically produced axioms. Its new version, OPPL 2 [12], provides a relatively rich pattern language, by allowing multiple variables in one script, and by aligning the OPPL syntax to the right level of abstraction. OPPL is based on OWL-API [10], which can be directly used for ontology manipulation, but OPPL makes this easier. On the other hand, OPPL focuses on axiom-based transformations, while OWL-API enables all types of axiom- and entity based transformations. Similarly to OPPL but in the context of Prolog, *POPL* (Prolog Ontology Processing Language) accessible via POSH (The Prolog OWL Shell) [17] envelopes the functionality of the Thea library [30]. A user of POPL can directly take advantage of the expressive power of the Prolog language for manipulating an ontology in terms of rewriting/adding axioms and refactoring ontology according to complex structural patterns. A simple rule (i.e. transformation pattern) for making all subclass-siblings disjoint, is

```
add X\~Y where (subClassOf(X,A),subClassOf(Y,A),X\=Y).
```

The *EvoPat* project [20] attempted to handle the transformation at both ontology and instance data levels within the scope of SPARQL (using both SELECT and UPDATE). Although its main declared focus was ‘repairing bad smells’ (i.e.

⁹<http://www.w3.org/TR/owl2-manchester-syntax/>

removal-centric transformation), it seems usable for a wide scope of transformations including style-level ones. However, due to its RDF-triple granularity, expressing OWL-level patterns in it requires more (and less comprehensible) code than in approaches operating at OWL level, such as OPPL or PatOMat. The approach by Lösch et al. [15] is similar in relying on SPARQL and primarily dealing with instance level of the ontology; however, the goal is to automatically apply change patterns capturing the evolution of the domain of interest, i.e. it truly supports *evolution* rather than transformation in the sense declared in this chapter.

As regards the *style transformation* of an OWL ontology to a simpler but OWL-compliant shape, an important task is translation to the SKOS vocabulary,¹⁰ which is capable of preserving its basic taxonomic structure as well as some ‘horizontal’ relationships among related concepts. A tool¹¹ for transforming OWL ontologies to SKOS classifications has recently been developed by Abdul Manaf in her thesis.¹² A similar project, which, in addition to OWL2SKOS transformation also supports improvement, enrichment and validation¹³ of SKOS classifications themselves, is Skosify [22].

12.5. Ontology Transformation using PatOMat Tools

In this section we present selected outcomes of the *PatOMat* project.¹⁴ We describe the structure of the transformation patterns employed, present the overall workflow and some implementation details of the PatOMat framework, the ‘ecosystem’ of collaborating tools (with special focus on the graphical user interface GUIPOT) and several use cases.

12.5.1. Transformation Pattern Representation

The central notion in the *PatOMat* transformation framework [34] is that of *transformation pattern* (TP). A TP contains two *ontology patterns*—the source OP and the target OP—and the description of the transformation between them, called *pattern transformation* (PT). The representation of *ontology patterns* is based on OWL 2 Manchester syntax.¹⁵ However, while an OWL ontology merely refers to particular entities, e.g., to the class *Person*, in the patterns we can refer both to particular entities and to their *placeholders* (denoted with a question mark at the beginning, e.g., *?OP1_A*). Placeholders are bound at the time of instantiation of a pattern.

Ontology Pattern An *ontology pattern* is a triple $\langle E, Ax, NDP^* \rangle$, such that E is

¹⁰<https://www.w3.org/2004/02/skos/>

¹¹<http://mowl-power.cs.man.ac.uk:8080/owl2skos/>

¹²Available from <https://www.escholar.manchester.ac.uk/uk-ac-man-scw:260745>.

¹³See <http://seco.cs.aalto.fi/tools/skosify/>

¹⁴The generic part of it had been funded by the Czech Science Foundation under no. P202/10/1825 (2010–2012), and the specific applications also by further projects, especially EU LOD2 (FP7 ICT 257943).

¹⁵<http://www.w3.org/TR/owl2-manchester-syntax/>

a non-empty set of *entity declarations*, Ax a (possibly empty) set of *axioms*, and NDP^* a (possibly empty) set of *naming detection patterns*.

*Entity declarations*¹⁶ concern *entity types* such as classes, properties and individuals either at the level of specified entities or at the level of placeholders. Object, data or annotation properties are considered. Annotation properties allow us to capture information about those parts of an ontology pattern which are not related to the logical meaning of the ontology. *Axioms* are statements about entities included in the transformation. Axioms can be either *optional* or *mandatory*. The latter are used for detection. Finally, the *naming detection pattern/s* capture the naming aspect of the ontology pattern relevant for its detection (i.e., it is not used if the pattern appears in the ‘target’ role).

Pattern Transformation Let OP1 and OP2 be ontology patterns. A *pattern transformation* from OP1 (source pattern) to OP2 (target pattern) is a tuple $\langle \text{LI}, \text{NTP}^* \rangle$, in which LI is a non-empty set of *transformation links*, and NTP^* is a (possibly empty) set of *naming transformation patterns*. Every *transformation link* $\text{l} \in \text{LI}$ is a triple $\langle \text{e}, \text{e}', \text{R} \rangle$ where $\text{e} \in \text{OP1}$, $\text{e}' \in \text{OP2}$, and R is either a *logical equivalence relationship* between *homogeneous* entities or an *extralogical relationship* between *heterogeneous* entities.

As *logical equivalence relationships* (\equiv) we consider standard OWL constructs declaring the equivalence/identity of two ‘logical entities’ of the same type: `owl:equivalentClass`, `owl:equivalentProperty` and `owl:sameAs` (for classes, properties and individuals, respectively). An *extralogical relationship* (\cong) can be 1) a relationship of type `eqAnn`, valid between a ‘logical’ entity and an annotation entity,¹⁷ or, 2) a ‘heterogeneous’ relationships `eqHet`, valid between two ‘logical entities’ of different type. Extralogical relationships correspond to ‘modeling the same real-world notion’ in different ways. *Naming transformation patterns* provide a way of giving names to (newly created) entities in OP2 with regard to entities in OP1 .

Transformation Pattern *Transformation Pattern* is a triple $\langle \text{OP1}, \text{PT}, \text{OP2} \rangle$ such that OP1 , OP2 are ontology patterns and PT is a pattern transformation from OP1 to OP2 .

Naming operations can be divided into *passive* ones, applied for checking purposes, and *active* ones, for naming a new entity.¹⁸ While both can be plugged into naming transformation patterns, only passive operations can be used in naming detection patterns.

We distinguish between two kinds of naming patterns:

- naming detection patterns (NDP), and
- naming transformation patterns (NTP).

¹⁶Corresponding to axioms with the `rdf:type` property.

¹⁷OWL 2 annotations may contain various interlinked entities in a separate ‘space’; these are, however, excluded from the logical interpretation of the ontology.

¹⁸A passive naming operation often has its active variant.

A *naming detection pattern* is a set of passive naming operations, $NDP = \{no_1, no_2, \dots, no_n\}$. All no_i have as their operands entities from the ontology pattern to which NDP belongs, and constants. As an example of an NDP with two operations we can take the following:¹⁹

```
{comparison(head_noun(?OP1_P), head_noun(?OP1_A), not_hypernym)}
```

This NDP checks whether or not there is a hypernym relationship present between two terms. In this case, those two terms are head nouns of entities (?OP1_P and ?OP1_A). For instance, considering ‘ScientificWork’ as ?A and ‘Paper’ as ?P then the pattern succeeds because ‘Paper’ is not a hypernym of ‘Work’ (applying WordNet [6]).

A *naming transformation pattern* is a set of pairs consisting of an entity and a naming operation, $NTP = \{(e_1, no_1), (e_2, no_2), \dots, (e_n, no_n)\}$. All no_i have as their operands entities from the source OP of the PT to which NTP belongs, and constants. All e_i are from the target OP of the PT to which NTP belongs. The following is an example of NTP with one compound operation:

```
{( ?G, make_passive_verb(?C) + head_noun(?A))}
```

This NTP names ?G as a compound of a verb’s passive variant in the head nouns of ?C and head noun of ?A. For instance, if ?A is bound with ‘PresentedPaper’ (with ‘Paper’ as the head noun) and ?C with ‘Rejection’ (with ‘Rejected’ as a passive verb form), the name of entity ?G in the transformed ontology will become ‘RejectedPaper’.

A small collection of *implemented naming operations* was gathered based on the requirements of use cases. They include (we list together both a passive and an active variant where relevant):

- *delimiter* detection and change (e.g., underscore or camel-case)
- detection and derivation of the *verb form* of a noun (using the ‘derivationally related forms’ resource from WordNet and the Stanford part-of-speech tagger [29])
- detection of *hypernym* relationship based on WordNet
- detection of *head noun* or its complement, for a noun phrase, and of a *head term* for a verb phrase, typically in a property name (only passive operation)
- construction of the *passive form* of a verb.

The individual *transformation operations* considered in PatOMat are of the following type: axiom addition, axiom removal, entity addition, entity removal and entity renaming. The strategies governing how a complex transformation defined by a transformation pattern boils down to such operations is described in detail in an overview paper [34].

12.5.2. Ontology Transformation Workflow in PatOMat

PatOMat-based transformation runs in a three-step workflow depicted in Fig. 12.1.

¹⁹This example is borrowed from transformation pattern available at: http://nb.vse.cz/~svabo/patomat/tp/np-new/tp_np1c.xml.

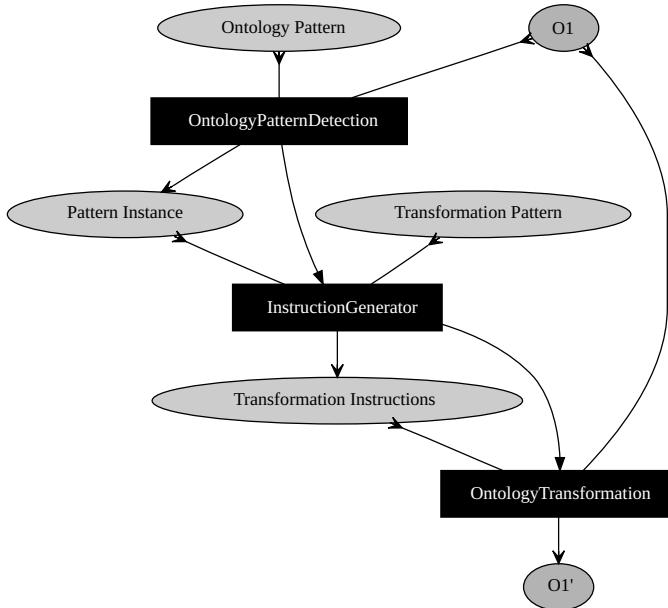


Figure 12.1. Ontology transformation workflow; application workflow is depicted using a line with a normal head and dataflow is depicted using a line with a V-shaped head

The *OntologyPatternDetection* service outputs the binding of entity placeholders. It takes the transformation pattern,²⁰ containing the source and target patterns, and a particular original ontology, on input. The service automatically generates a *SPARQL query*²¹ internally, based on the mandatory part of the ontology pattern (the placeholders becoming SPARQL variables) and executes it. The structural/logical aspect is captured in the query structure, and the possible *naming constraint* is specifically dealt with based on its description within the source pattern. The implementation uses the Jena framework.²² The *InstructionGenerator* service outputs particular transformation instructions, also in XML. It takes the particular binding of placeholders and the transformation pattern on input. Transformation instructions are generated according to the transformation pattern and the pattern instance. The *OntologyTransformation* service outputs the transformed ontology. It takes the particular transformation instructions and the particular original ontology on input.

The intermediate products, pattern instance and transformation instructions, are assumed to be inspected and possibly edited by the user. In particular, the user can choose which pattern instances (from automatic detection) should be used. The transformation framework allows to log information about added and/or removed axioms to ontology annotations.

²⁰Transformation patterns are serialised according to an XML schema: <http://nb.vse.cz/~svabot/patamat/tp/tp-schema.xsd>

²¹<http://www.w3.org/TR/rdf-sparql-query/>

²²<http://jena.apache.org/>

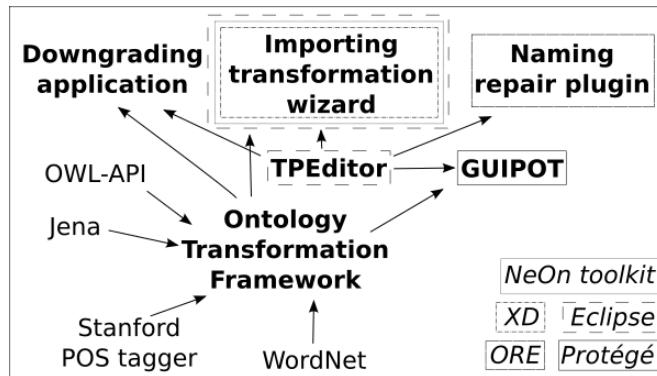


Figure 12.2. Ontology transformation framework with tools it either reuses or to which it is integrated

12.5.3. PatOMat Technological Architecture Overview

The transformation framework employs several semantic web or other tools and already has support in several user interfaces. These are depicted in Figure 12.2; our tools are written in bold font. The first group of tools (mostly on the left-hand side in Figure 12.2) are libraries that the transformation framework employs. The *OWL-API*²³ allows the framework to add/remove axioms (including annotations) and to add/remove entities and it is used for execution of transformations. The *Jena semantic web framework*²⁴ is used in the detection step in order to perform SPARQL querying by its *ARQ* component.²⁵ Regarding the naming aspect, on the one hand we employ the *Stanford part-of-speech tagger*²⁶ in order to get the part-of-speech for particular tokens of entities. On the other hand we use *WordNet*²⁷ in order to get hypernyms, meronyms or to create the verbalised form of a noun.

The second group of tools deals with user interfaces [23]. *Transformation Pattern Editor* (TPEditor) supports authoring and updating of transformation patterns. It allows their graphical modelling and export/import from/to the (XML-based) TP notation. TPEditor is available as a plugin for Eclipse and uses the Graphical Editing Framework.²⁸ *Graphical User Interface for Pattern-based Ontology Transformation* (GUIPOT) is a Protégé plugin allowing the user to go through all steps of transformation via a standard working environment of a knowledge engineer. We devote a special subsection to it below. The *Importing transformation wizard* supports the *best-practice artifact inclusion* use case in the sense that the user can select a content ontology design pattern, an ontology and a transformation pattern, and a specific process will be performed, see Sec-

²³<http://owlapi.sourceforge.net/>

²⁴<http://jena.sourceforge.net/>

²⁵<http://jena.apache.org/documentation/query/index.html>

²⁶<http://nlp.stanford.edu/software/tagger.shtml>

²⁷<http://wordnet.princeton.edu/>

²⁸<http://www.eclipse.org/gef/>

tion 12.5.5. This wizard is integrated into the *eXtreme Design* tool supporting pattern-based ontology development and can be plugged²⁹ into Eclipse as well as into the NeOn toolkit.³⁰ The *Naming repair plugin* supports the naming repair use case. It is integrated into the *Ontology Repair and Enrichment* toolkit (ORE)³¹ [14]. Finally, there is a web-based *Downgrading application* supporting the language profiling use case.³² Following the input of the source ontology URI (and selected TP in the ‘one construct transformation’ use case), the transformed ontology is displayed (together with a brief transformation log) and a link to its code is also provided.

12.5.4. GUIPOT: Protégé plugin for PatOMat-Powered Transformation Pattern Application

The main functionality of GUIPOT is visualizing the ontology before the transformation, allowing the user to select a desired subset of detected pattern instances to be transformed and visualizing the ontology after the transformation, highlighting the changes.

GUIPOT allows the user to load a transformation pattern (created either manually or using another tool) and display a list of pattern instances detected in the given ontology. If one or more instances from the list are selected, they are highlighted in a classical Protégé hierarchy view and also visualized in a node-link view on the left part of the plugin window (Figure 12.4). The visualization can be switched to four different levels of detail. The default one uses OntoGraf³³ and displays classes as nodes with properties as links between them (according to domain/range relationships). A more detailed view, where properties and complex relationships between classes (e.g., complementOf) are displayed as separate nodes, is implemented using SOVA³⁴ visualization plugin. SOVA is also used for less detailed overviews of class hierarchy where the class nodes are laid out with spring layout or tree layout algorithms. The detailed views show fragments of the ontology involved in the selected pattern instances. The overviews show the whole ontology with entities involved in the selected pattern instances highlighted.

When the user is satisfied with the selection of pattern instances, s/he just clicks the “Apply Transformation” button. The right part of the window shows the ontology after transformation, with affected entities indicated by red arrows and highlighted or focused in the visualization (analogically to the visualization of selected pattern instances). The transformed ontology can then be loaded into Protégé with a single click and then either edited manually or transformed with another pattern.

²⁹Information about these front-end tools are available at <http://owl.vse.cz:8080/tools.html>

³⁰http://neon-toolkit.org/wiki/Main_Page

³¹<http://ore.aksw.org/ore>

³²Available from <http://owl.vse.cz:8080/Downgrading/>.

³³<http://protegewiki.stanford.edu/wiki/OntoGraf>

³⁴<http://protegewiki.stanford.edu/wiki/SOVA>

12.5.5. Overview of PatOMat Use Cases

We subsequently present four use cases, roughly corresponding to the transformation motivations from the beginning of Section 12.4.

Language profiling and downgrading Different sub-languages of OWL have specific restrictions on the (combinations of) allowed constructs. For tweaking the ontological representation so as to comply with these restrictions, transformation patterns can be used. In PatOMat we experimented both with explicit replacement of a specific *construct* and with transformation to a representation complying with an OWL *profile*. An example³⁵ of disallowed class, for the OWL EL profile, is that of enumerated class with more than one individual. The axiom

```
EurAsia = {europe, asia}
```

then needs to be transformed by something like

```
Europe_nc = { europe }. Asia_nc = { asia }.
Europe_nc SubClassOf: EurAsia
Asia_nc SubClassOf: EurAsia
```

The strong point of the pattern-based approach, in contrast to research focused on logical inference, is that the users can easily design their own transformation patterns to address a certain, specific and unforeseen, construct-replacing use case or complexity downgrading for a certain, newly introduced profile. If such patterns are shared, other users could easily apply them through the on-line transformation web services (i.e. without the necessity to install a particular reasoner as in the logic-centric approaches to transformation).

Experiments have also been carried out with *SKOSification* of OWL ontologies. This also, in a way, corresponds to the downgrading scenario, albeit with more radical impact on the ontology structure.

Readability improvement and anti-pattern elimination It is quite common in ontologies that a subclass has the *same head noun* as its parent class.³⁶ By an earlier study [25] we estimate that in ontologies for technical domains this simple pattern is verified in 50–80% of class-subclass pairs such that the subclass name is a *multi-token* one. This number further increases if we consider *thesaurus correspondence* (synonymy and hypernymy) rather than literal string equality. In fact, the set-theoretic nature of taxonomic path entails that the correspondence of head nouns along this path should be close to 100% in principle; the only completely innocent deviations from it should be those caused by incomplete thesauri. In other words, any violation of head noun correspondence may potentially indicate a (smaller or greater) problem in the ontology and can be viewed as *anti-pattern*. Prototypical situations are:

³⁵For more detail and further examples see [24].

³⁶The head noun is typically the last token, but not always, in particular due to possible prepositional constructions, as, e.g., in ‘HeadOfDepartment’.

- Inadequate use of class-subclass relationship, typically in the place of whole-part or class-instance relationship, i.e., a *conceptualisation error* frequently occurring in novice ontologies.
- *Name shorthanding*, typically manifested by use of adjective, such as ‘StateOwned’ (subclass of ‘Company’).

While the former requires complex refactoring of the ontology fragment, the latter – which is mainly a readability issue, especially when visualizing complex axioms – can be healed by propagation of the parent name down to the child name.

We implemented support for the parent name propagation, in the form of transformation patterns, into a version of PatOMat service integrated into the ORE (Ontology Repair and Enrichment) tool [14] developed by AKSW, University of Leipzig. The transformation process is visualized in a single view as shown in Figure 12.3. Here the user can select a naming pattern, as for example **non-matching child1** (corresponding to the *St* variant of the non-matching child pattern), in the leftmost list. PatOMat then detects instances of the selected pattern in the processed ontology, e.g., `[?OP1_P=Contribution; ?OP1_A=Poster]`. For the selected pattern instances the user will be provided a list of renaming instructions, for example to rename the class **Poster** to **PosterContribution**, which can then be used to transform the ontology and solve the detected naming issues. An empirical study has been carried out in applying this tool on ontologies from the LOV portal [33], revealing a number of interesting semantic and syntactic issues in ontology class naming.

Best-practice artifact inclusion As best-practice artifact (BPA) we understand either a fully-blown ontology (foundational one or core one within a domain) or an ontology content pattern as a smaller, widely usable module. When such a BPA is to be integrated into a legacy ontology – typically to its root portion – and their encoding styles differ, adaptation may be needed.

We performed studies for adapting ontologies both to content patterns such as AgentRole [28] and to a core ontology, which, in this case, was GoodRelations [9] (GR), the highly popular general ontology of the e-commerce domain. In the former case, ontologies describing the domain of organizing conferences³⁷ have been overhauled so as to comply with the AgentRole encoding style, having explicit classes for roles played by people or other agents; these roles were derived either from classes of persons or with object properties having a class of persons in domain. In the latter case, the legacy ontologies were various specialized product ontologies, for example, extracted from Freebase [2]. Compared to the former case, the changes to the ontology structure were mostly less dramatic; they consisted, e.g., in subordinating the properties expressing the product parameters to abstract GR properties such as *gr:qualitativeProductOrServiceProperty*, or changing Boolean properties to subclasses of a ‘feature’ class. The result of such an adaptation is shown in the previously introduced screenshot of GUIPOT, Fig. 12.4. The source pattern instantiation selected in the upper-middle pane corresponds to a Boolean-valued property *digicams.digital_camera.orientation_sensor*, from which an equally-named individual of class *Feature* (in the bottom-middle-right

³⁷<http://owl.vse.cz:8080/ontofarm/>

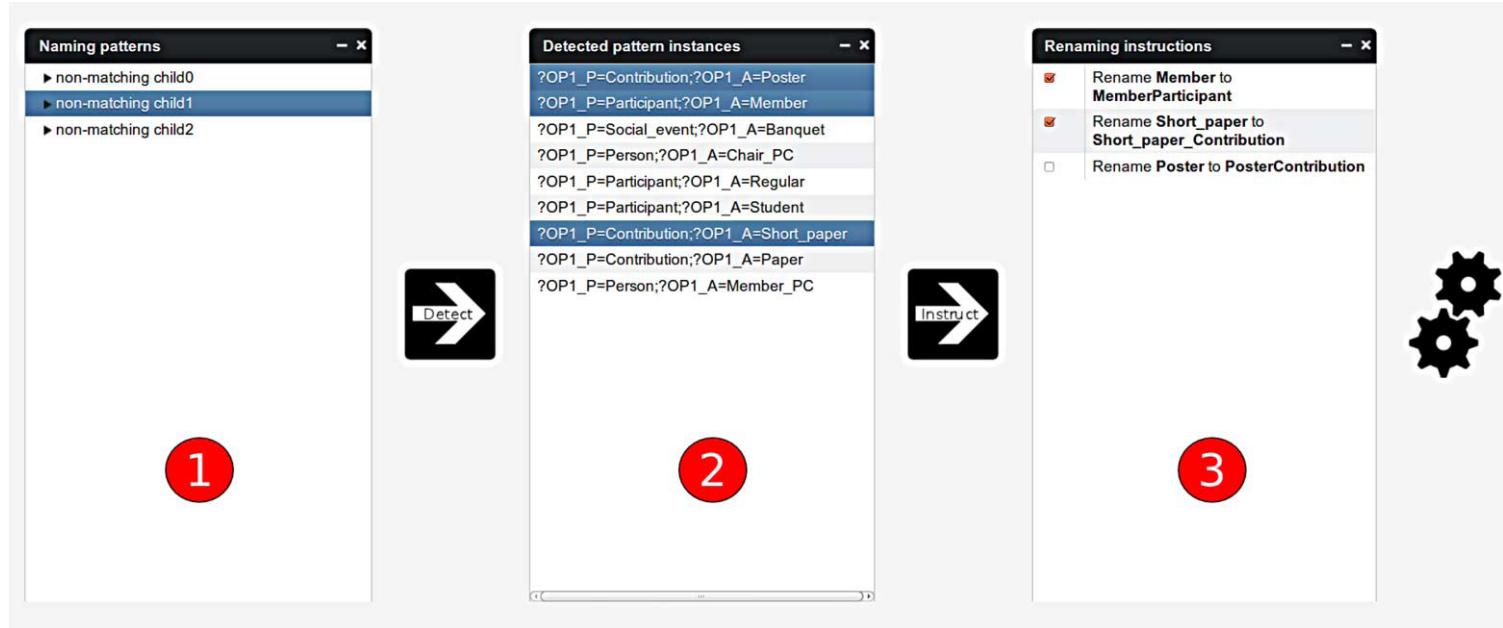


Figure 12.3. Screenshot of the PatOMat view in the ORE tool.

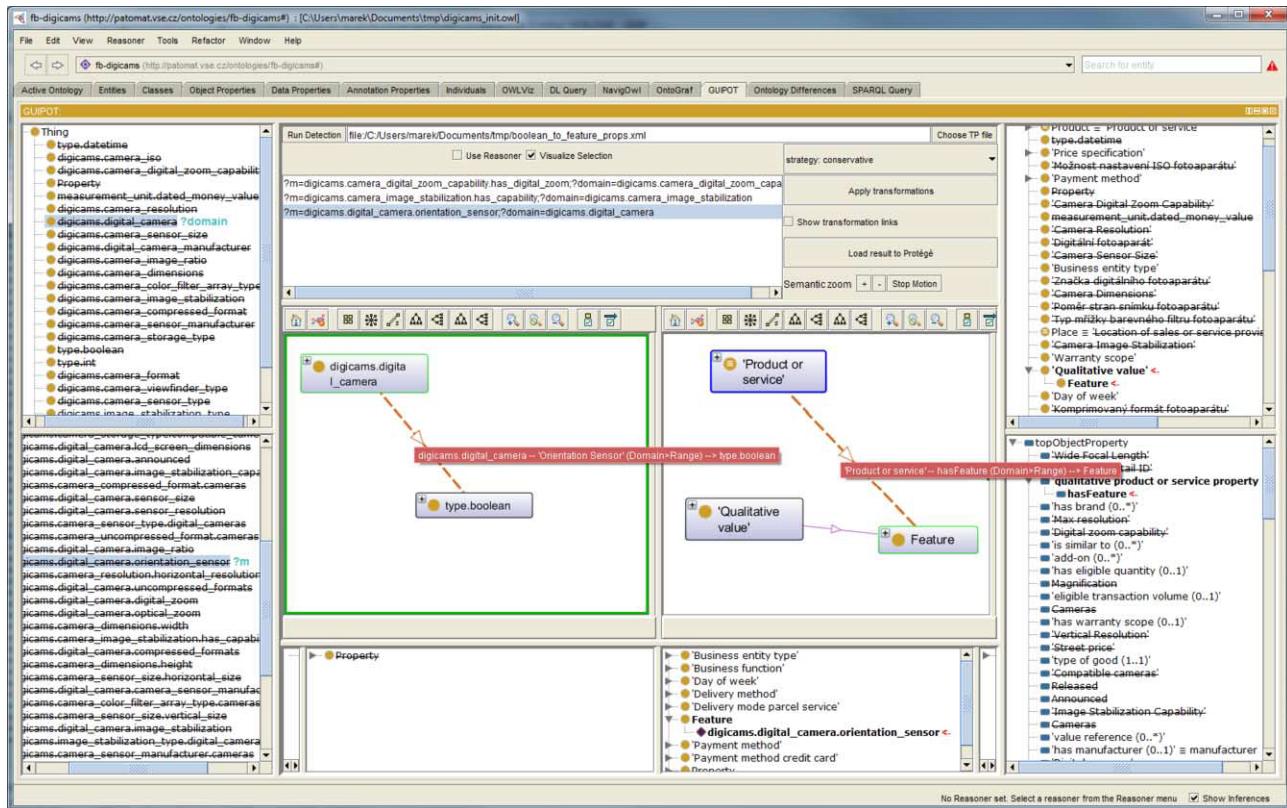


Figure 12.4. GUIPOT interface after application of transformation: adaptation of product ontology for GoodRelations

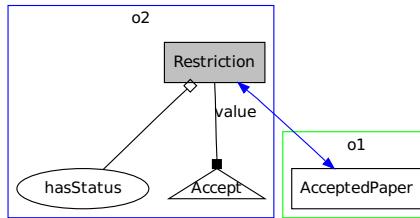


Figure 12.5. Instance of alignment pattern ‘Class By Attribute Value’

pane) is generated.

Ontology matching Most ontology matching (OM) tools deliver simple entity-to-entity correspondences. Complex matching can be mediated by alignment (originally called “correspondence”) patterns [21], which however most OM tools do not support. Attempting to transform, prior to matching, an ontology to its variant using transformation patterns, could thus help the OM tools.

We can imagine the situation³⁸ where in one conference-organization ontology the notion of ‘accepted paper’ is formalized through an existential restriction while in the other ontology it is a named class. This corresponds to the ‘Class By Attribute Value’ alignment pattern, depicted in Figure 12.5.

Based on this alignment pattern, a transformation pattern called *tp-hasValue* can be defined as follows:

OP1 : $E = \{\text{Class: } A, \text{ ObjectProperty: } p, \text{ Individual: } a\}, Ax = \{A \text{ EquivalentTo: } (p \text{ values } a)\}$,

OP2 : $E = \{\text{Class: } B, \text{ Literal: } An1, \text{ Literal: } An2\}, Ax2 = \{B \text{ annotation : } \text{discr_property } An1, B \text{ annotation:value } An2\}$,

PT : $LI = \{A \text{ EquivalentTo: } B, p \text{ eqAnn: } An1, a \text{ eqAnn: } An2\}, \text{enp}(B) = \text{make_passive_verb}(a) + \text{head_noun}(A)$,

enp(B) is an example of entity naming transformation pattern, which enables to make a proper name for a new entity, e.g. from ‘Accept’ as *a* and ‘PresentedPaper’ as *A* it makes ‘AcceptedPaper’ as *B*.

A particular instantiation of this pattern could be as it follows: *O1* : $A = \text{PresentedPaper}, p = \text{hasStatus}, a = \text{Accept}$ *O2* : $B = \text{AcceptedPaper}, An1 = \text{'hasStatus'}, An2 = \text{'Accept'}$, see Figure 12.6.

By applying the *tp-hasValue* pattern on *O1* we can get a new entity ‘AcceptedPaper’ in *O1'*, which is a perfect match for entity ‘AcceptedPaper’ from *O2* by any simple string-based technique.

³⁸While the ontology matching use case of transformation patterns is potentially quite interesting, no real-world study has however been carried out to date.

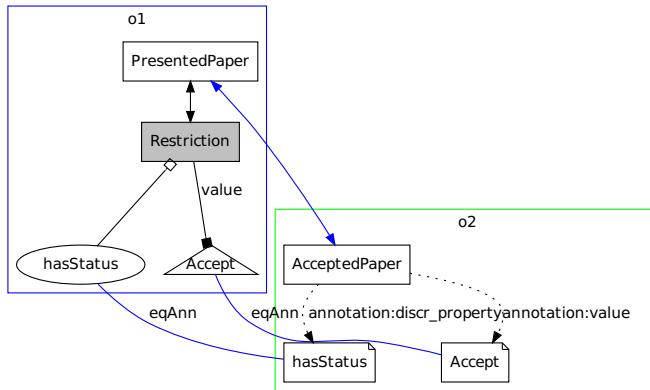


Figure 12.6. Instance of transformation pattern for `hasValue`

12.6. Conclusions

The need for transformation is omnipresent in the world of ontological modeling. We first distinguished among the different types of transformation tasks and motivations, and then explained selected approaches, especially those of the OWL2OWL and OBM2OWL species, in more detail. As the field of ontology transformation matures, hardcoded first-choice methods are gradually being replaced by pattern-based approaches that provide more flexibility and can thus cater for the different needs arising in different use cases and domains.

By the practical experience gained from the different use cases (ours or those reported by other researchers), we assume that OWL2OWL transformation is viable provided all or most of the following conditions are satisfied:

1. the ontologies to be transformed are medium-complex and at least medium-sized;
2. there are several of them sharing similar characteristics
3. there are coherent (even if suboptimal) naming conventions used.

The reason is that for small and solitary ontologies the overhead associated with transformation pattern design might not pay off compared to entirely manual rewrite of the ontologies; on the other hand, for trivially-structured (albeit large) ontologies it might not pay off for the knowledge engineer to learn the transformation pattern language, since a simple script can do the business. Since an inherent part of the transformation pattern language is its naming-pattern-handling component, naming conventions allow the detection components to limit the amount of input pattern occurrences and thus keep the manual intervention of the knowledge engineers in tolerable limits; furthermore, the possibility of transforming the

naming along the logical structure (towards best-practice naming) is an obvious added value.

The more recently introduced OBM2OWL transformation species indicates great potential as fuel for large-scale reuse of ontological conceptualizations across domains and OWL encoding styles. On the other hand, the need to learn another ontological formalism beyond OWL might be an obstacle for casual knowledge engineer. Instructive support by visualization tools and emphasis on concrete examples (as offered, for instance, by the PURO approach), may however alleviate this problem.

The research underlying this chapter has/had been mainly supported by CSF P202/10/1825 (PatOMat, 2010–2012), FP7 ICT 257943 (LOD2, 2011–2014), CSF 14-14076P (COSOL, 2014–2016) and VSE IGA F4/28/2016.

Bibliography

- [1] P. P. F. Barcelos, V. A. dos Santos, F. B. Silva, M. E. Monteiro, and A. S. Garcia. An automated transformation from OntoUML to OWL and SWRL. In M. P. Bax, M. B. Almeida, and R. Wassermann, editors, *ONTOBRAS*, volume 1041 of *CEUR Workshop Proceedings*, pages 130–141. CEUR-WS.org, 2013.
- [2] M. Dudáš, O. Zamazal, J. Mynarz, and V. Svátek. Exploiting Freebase to obtain GoodRelations-based product ontologies. In M. Hepp and Y. Hoffner, editors, *EC-Web*, volume 188 of *Lecture Notes in Business Information Processing*, pages 34–45. Springer, 2014.
- [3] M. Dudáš, T. Hanzal, V. Svátek, and O. Zamazal. OBM2OWL patterns: Spotlight on OWL modeling versatility. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *WOP*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [4] M. Dudáš, T. Hanzal, V. Svátek, and O. Zamazal. OBOWLmorph: Starting ontology development from PURO background models. In *OWLED*, volume 9557 of *LNCS*. Springer, 2015.
- [5] M. Egana, A. L. Rector, R. Stevens, and E. Antezana. Applying ontology design patterns in bio-ontologies. In A. Gangemi and J. Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 7–16. Springer, 2008.
- [6] C. Fellbaum, editor. *Wordnet, an Electronic Lexical Database*. MIT Press, 1998.
- [7] J. Guerson, T. P. Sales, G. Guizzardi, and J. P. A. Almeida. OntoUML lightweight editor: A model-based environment to build, evaluate and implement reference ontologies. In J. Kolb, B. Weber, S. Hallé, W. Mayer, A. K. Ghose, and G. Grossmann, editors, *EDOC Workshops*, pages 144–147. IEEE Computer Society, 2015.
- [8] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. Number 15 in Telematica Institute Fundamental Research Series. Telematica Instituut, Enschede, The Netherlands, 2005.

- [9] M. Hepp. Goodrelations: An ontology for describing products and services offers on the web. In A. Gangemi and J. Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2008.
- [10] M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.
- [11] L. Iannone, M. E. Aranguren, A. L. Rector, and R. Stevens. Augmenting the expressivity of the Ontology Pre-Processor Language. In C. Dolbear, A. Ruttenberg, and U. Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [12] L. Iannone, A. L. Rector, and R. Stevens. Embedding knowledge patterns into OWL. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. P. B. Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2009.
- [13] K. Janowicz, P. Hitzler, B. Adams, D. Kolas, and C. Vardeman. Five stars of linked data vocabulary use. *Semantic Web*, 5(3):173–176, 2014.
- [14] J. Lehmann and L. Bühmann. ORE – a tool for repairing and enriching knowledge bases. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Z. 0007, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *International Semantic Web Conference (2)*, volume 6497 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2010.
- [15] U. Lösch, S. Rudolph, D. Vrandečić, and R. Studer. Tempus fugit - towards an ontology update language. In L. A. et al., editor, *6th European Semantic Web Conference (ESWC 09)*, volume 5554 of *Lecture Notes on Computer Science*, pages 278–292. Springer-Verlag, Juni 2009.
- [16] E. Motta. An overview of the OCML modelling language, 1998. 8 th Workshop on Knowledge Engineering: Methods & Languages KEML 98.
- [17] C. Mungall. Posh - the Prolog OWL Shell. In M. Dumontier and M. Courtot, editors, *OWLED*, volume 796 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
- [18] N. Noy. Representing classes as property values on the semantic web. W3C Working Group Note, April 2005.
- [19] R. Palma, F. Zablith, P. Haase, and O. Corcho. Ontology evolution. In M. C. Suarez-Figueroa, A. Gomez-Perez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World*, pages 235–255. Springer, 2012.
- [20] C. Riess, N. Heino, S. Tramp, and S. Auer. EvoPat – pattern-based evolution and refactoring of RDF knowledge bases. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 647–662. Springer, 2010.
- [21] F. Scharffe, O. Zamazal, and D. Fensel. Ontology alignment design patterns. *Knowl. Inf. Syst.*, 40(1):1–28, 2014.
- [22] O. Suominen and E. Hyvönen. Improving the quality of SKOS vocabularies with Skosify. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d’Aquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *EKAW*, volume 7603 of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2012.

- [23] O. Šváb-Zamazal, M. Dudáš, and V. Svátek. User-friendly pattern-based transformation of owl ontologies. In A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Aquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, editors, *EKAW*, volume 7603 of *Lecture Notes in Computer Science*, pages 426–429. Springer, 2012.
- [24] O. Šváb-Zamazal, A. Schlicht, H. Stuckenschmidt, and V. Svátek. Constructs replacing and complexity downgrading via a generic OWL ontology transformation framework. In *SOFSEM*, pages 528–539, 2013.
- [25] O. Šváb-Zamazal and V. Svátek. Analysing ontological structures through name pattern tracking. In A. Gangemi and J. Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2008.
- [26] V. Svátek, M. Homola, J. Kluka, and M. Vacura. Mapping structural design patterns in OWL to ontological background models. In V. R. Benjamins, M. d'Aquin, and A. Gordon, editors, *K-CAP*, pages 117–120. ACM, 2013.
- [27] V. Svátek, M. Homola, J. Kluka, and M. Vacura. Metamodeling-based coherence checking of OWL vocabulary background models. In M. Rodriguez-Muro, S. Jupp, and K. Srinivas, editors, *OWLED*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
- [28] V. Svátek, O. Šváb-Zamazal, and M. Vacura. Adapting ontologies to content patterns using transformation patterns. In E. Blomqvist, V. K. Chaudhri, O. Corcho, V. Presutti, and K. Sandkuhl, editors, *WOP*, volume 671 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
- [29] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, 2003.
- [30] V. Vassiliadis, J. Wilemaker, and C. Mungall. Processing OWL2 ontologies using Thea: An application of logic programming. In R. Hoekstra and P. F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [31] B. Villazon-Terrazas and A. Gomez-Perez. Reusing and re-engineering non-ontological resources for building ontologies. In M. C. Suarez-Figueroa, A. Gomez-Perez, E. Motta, and A. Gangemi, editors, *Ontology Engineering in a Networked World*, pages 107–145. Springer, 2012.
- [32] W3C. RDF 1.1 Concepts and Abstract Syntax, 2014. <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.
- [33] O. Zamazal, L. Bühmann, and V. Svátek. Checking and repairing ontological naming patterns using ORE and PatOMat. In P. Lambrix, G. Qi, M. Horridge, and B. Parsia, editors, *WoDOOM*, volume 999 of *CEUR Workshop Proceedings*, pages 69–76. CEUR-WS.org, 2013.
- [34] O. Zamazal and V. Svátek. PatOMat - versatile framework for pattern-based ontology transformation. *Computing and Informatics*, 34(2):305–336, 2015.
- [35] V. Zamborlini and G. Guizzardi. On the representation of temporally changing information in OWL. In *EDOCW*, pages 283–292. IEEE Computer Society, 2010.

Chapter 13

Ontology Design Patterns for Data Integration: The GeoLink Experience

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

13.1. Introduction

Anybody who works with multiple data sources would know that *data integration* [15], i.e., accessing data from those multiple sources via a unified view is a major, practical challenge. There are at least three issues causing the difficulties in data integration [3, Section 1.2]. The first is concerned with the lack of syntactic and architectural interoperability between the systems involved in a data integration effort due to the heterogeneity in data format, architectural constraints and configurations, access method, and query processing capabilities. The second has to do with semantic heterogeneity because each data source has its own independently developed schema leading to differences in vocabularies, levels of granularity in data model, or conceptualization. Meanwhile, the third is rather non-technical in nature and it boils down to inability or unwillingness of the data providers to fully participate in a data integration effort because of the worry that participating in the integration may entail major changes in the established management practices and policies [21].

What we need is thus a data integration framework that is *flexible* and *extendible*. The former implies that data sources do not have to make drastic changes in their local schema design, nor do they have to significantly alter their established business processes just to participate in the integration, while the latter means that joining (and also, leaving) the integration does not necessitate costly changes to the whole framework. Berners Lee, et al [1] implicitly suggested that such a framework can be realized if we put an emphasis on *linking* data and the use of *ontologies*.

Intuitively, Linked Data can help in addressing the syntactic interoperability problem, while ontologies can help bridge semantic heterogeneity. The intuition

was not always clear. In the beginning, there was a tacit assumption that extensively axiomatized, monolithic ontologies would be the only everything we need [16]. However, people seemed to later realize that such ontologies are hard to build and typically make very specific ontological commitments, deeply ingrained in the complexity of the axiomatization. Data providers often find such ontologies difficult to understand and even when they can be understood, the ontological commitments may be too strict to be accepted by all parties involved in the integration. This realization then fueled the shift toward pure Linked Data that employs extremely lightweight and often semiformal vocabularies [4], spurring the growth of Web of Data [20]. It has been argued, however, that solely relying on Linked Data is not enough as such semiformal vocabularies only provide us with too little schema information, which is needed for understanding and integrating different linked datasets [7].

To tackle the above problems, it seems that a middle ground is needed to take the best of both worlds: use Linked Data together with ontologies that are a little bit more than just semiformal vocabularies, but the ontologies should neither be too extensively axiomatized nor monolithic. Furthermore, Ontology Design Patterns (ODPs) seem to fit the bill since they contain formalization strong enough that they go beyond semiformal vocabularies, while at the same time, are modular by design that they are neither monolithic nor contain too intricate axiomatization [17].

In this chapter, we describe our experience in using ODPs for the purpose of a flexible and extendible data integration framework involving multiple, semantically heterogeneous data repositories in the context of our recent GeoLink project in collaboration with several US-based ocean science data repositories. In particular, This chapter takes a bird-eye view to the approach we took from Chapter 10 [13] where the dual schema idea was used to cater multiple perspectives from the users when publishing Linked Data. Parts of the content of this chapter have been published elsewhere [12, 14] and we refer the reader to those prior publications for more details.

13.2. Overview of the GeoLink Project¹

Research in ocean science, very much like in other branches of science, relies very strongly on the availability of high quality data. A significant portion of the data is obtained from various field expeditions to the ocean, which is often expensive or difficult to repeat because one needs an access to an appropriate vessel and there is only limited number of such a vessel in the world. More importantly, synthesis centers such as NCEAS² and NESCent,³ have demonstrated compelling evidences that interdisciplinary analyses over existing data potentially lead to novel scientific insights [5, 6, 19]. As a result, funding agencies realized the importance of establishing data repositories to improve data preservation. Data repositories, such as BCO-DMO, DataONE, and IODP, were designed to serve

¹<http://www.geolink.org>; see also <http://schema.geolink.org> for the specification of the schema and <http://data.geolink.org> for a SPARQL endpoint

²<https://www.nceas.ucsb.edu/>

³<http://www.nescent.org>

a particular part of the geoscience research community in order to keep quality control and data management more manageable. Unfortunately, there are unintended consequences as data became highly heterogeneous due to differences in formats, access methods and nuances in conceptualization. From researchers' perspective, this is quite often frustrating as they have to find and integrate data across separate repositories when conducting an integrative analysis [23]. Motivated by these challenges, the National Science Foundation (NSF) launched the EarthCube program several years ago to upgrade cyberinfrastructure for the geosciences through community-led effort, which is realized in the form of building block projects, research coordination networks, and other activities aiming to enable cross-discipline data sharing and integration, allow global data discovery, and transform the way geoscience research is done in understanding the Earth system.

GeoLink is a building block project within EarthCube that specifically looking at how advances in semantic technologies can be effectively used to realize a data integration and discovery framework across multiple repositories where there is a lack of direct alignment between data from different repositories and differences in the way data and knowledge are modeled. As part of the partnerships, the following data repositories, mainly in ocean science, are involved: BCO-DMO,⁴ DataONE,⁵ IEDA,⁶ IODP,⁷ LTER,⁸ MBLWHOI Library,⁹ and R2R.¹⁰ This project employs Linked Data and ODPs, the former allows repositories publish their data in a standard syntax with links to data from other repositories, while the latter enables horizontal integration over independent semantic models.

13.3. The GeoLink Architecture

Standard data integration architecture, as described, e.g., by Doan, et al. [3] usually consists of multiple data sources and an integration layer between the users and the data sources containing the global schema. Users can query the data through an endpoint provided by the integration layer using vocabulary defined in the global schema without the need to understand exactly how the query on the data sources looks like. The integration layer can be realized as a virtual layer or a data warehouse. With the former, the integration layer does not store the data physically and only essentially performs a translation from the user's query into appropriate queries to the data sources, whereas with the latter, the data from all data sources are physically loaded into the integration layer after undergoing transformation to conform with the global schema.

In ontology-based data integration, the global schema is given in the form of an ontology. The use of ontology as a global schema for data integration has

⁴Biological and Chemical Oceanography Data Management Office, <http://www.bco-dmo.org/>

⁵Data Observation Network for Earth, <https://www.dataone.org/>

⁶Interdisciplinary Earth Data Alliance, <http://www.iedadata.org/>

⁷International Ocean Discovery Program, <http://www.iodp.org/>

⁸Long Term Ecological Research Network, <http://www.lternet.edu/>

⁹Marine Biological Laboratory/Woods Hole Oceanographic Institution Library, <http://www.mblwholibrary.org/>

¹⁰Rolling Deck to Repository Program, <http://www.rvdata.us/>

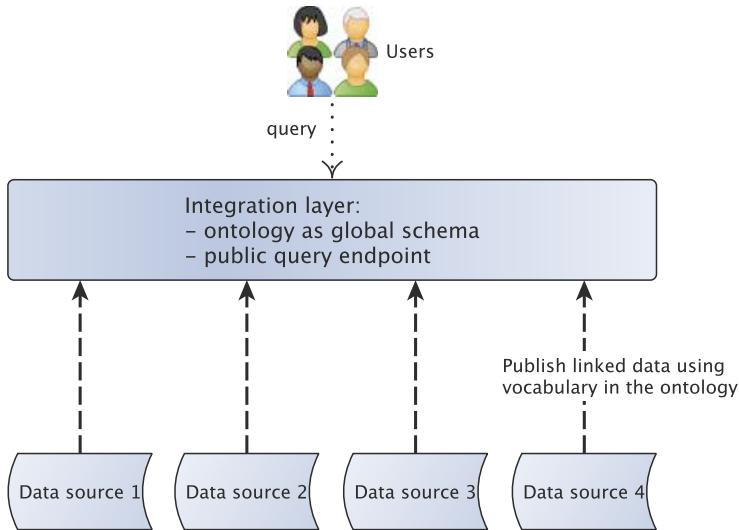


Figure 13.1. Ontology-based Linked Data Integration

been known prior to the popularity of Semantic Web and Linked Data [22]. The emergence of the latter [2], however, naturally led to the idea of *ontology-based linked data integration* where not only ontology is used as the global schema, but also the data from the participating data sources are exposed as linked data. A typical architecture for this kind of data integration is depicted in Fig. 13.1. In this scheme, a data source publishes its data as a collection of RDF triples while employing shared vocabulary defined in the ontology. The integration layer can either materialize this published data from all data sources into one location (a warehouse solution) or simply rewrite the query from the user into separate queries to the data sources. Note that this does not preclude a data source to simultaneously serve the data directly to the public on its own.

As mentioned in the introduction, one of the problems lies in the construction of the ontology as the global schema. To avoid the difficulties faced when complex, monolithic ontologies are used, the GeoLink project employs ontology design patterns (ODPs) to form a *modular global schema*. More precisely, we employ the so-called Content (Ontology Design) Patterns, which are domain dependent ODPs that are designed to provide definition of vocabulary in the domain of interest [18]. Content Patterns are realized as small ontologies that can be composed together as modules of a larger ontology, which we use here as the global schema. The project in principle goes through two major steps:

1. modeling key notions in the ocean science domain as Content Patterns (CPs), which then form the global schema;
2. populating the global schema with data according to the Linked Data principles.

13.3.1. Content Pattern Modeling

Modeling was conducted collaboratively involving ontology engineers and data providers who also act as domain experts. The process is done in a way similar to the one described by Janowicz [8]. Modularity is ensured by iteratively modeling one key notion at a time. That is, we pick one key notion to model and focus with fleshing out this notion until we get a CP. This key notion may of course need the presence of another key notion, which in this case is handled by providing a hook akin to what we described in Chapter 1 [11]. In a subsequent modeling iteration, the notion referred to by this hook can then be modeled if necessary. As a result of this process, we identified the following notions modeled as CPs, whose overview is depicted in Fig. 13.2.

Person, Organization, Agent, Agent Role. *Person* corresponds to human persons, while *Organization* represents organizations and institutions. Both are generalized as *Agent*, and furthermore, their involvement in events and other notions is modeled through the use of *Agent Role* pattern, which is a variant of the one discussed in Chapter 16 [10].

Event, Place, Geometry, Time Entity. *Event* represents the notion of event: an abstract entity that happens within some spatiotemporal extent or boundary. The introduction of *Event* itself actually happened when we modeled *Cruise* (see below), which was the first concrete, non-trivial ocean science notion that we considered. In this project, we think that separating *Place* (a spatial component) from *Time Entity* since this appears to be closer to the way the actual data is modeled. *Place* is a generic placeholder for information about places and separated from instances of *Geometry* as their spatial footprint. This allows one to consider places that are more abstract than specific locations with geo-coordinates.

Cruise, Vessel, Platform, Dive, Submersible. *Cruise* is a key notion in ocean science that is occurring in multiple data sources. In particular, both R2R and BCO-DMO are the main repositories that maintain data about the United States oceanographic cruises. *Cruise* is modeled as an event corresponding to a scientific expedition that is undertaken by a vessel traveling through some trajectories. Here, *Vessel* is modeled as a separate pattern. *Vessel* is later generalized into the notion of *Platform* and from this, the notion of *Dive* and *Submersible* can be modeled. *Dive* is similar to *Cruise* but undertaken by a submersible, which is a craft or boat that is designed to operate while submerged under the water surface.

Funding Award, Program. *Funding Award* represents grants that fund scientific activities. *Program* is meant for a named scientific effort, typically from some funding agency, that spans several funding awards.

Dataset, Digital Object, Document. *Dataset* represents a logical data collection unit with a particular scientific theme that may consist of several digital objects (e.g., files, images, etc.). *Digital Object* represents any one of the aforementioned digital objects. *Document* is similar to *Dataset*, but intended to represent entities such as papers, reports, theses, etc.

Physical Sample, Sampling Process. *Physical Sample* is a notion that corresponds to a logical unit of physical sample data that scientists collect from

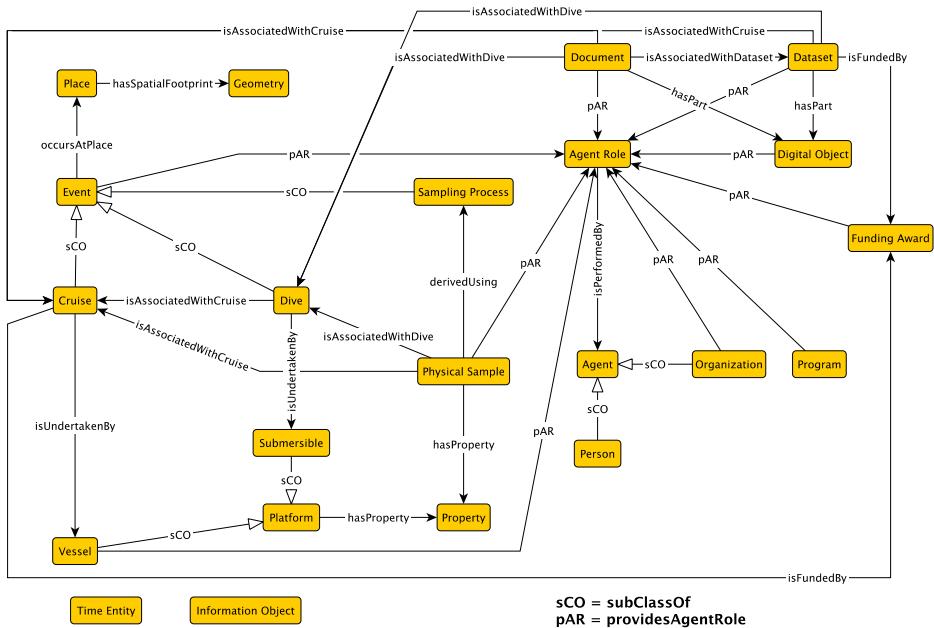


Figure 13.2. Overview of the GeoLink content patterns and the main links among them – all patterns have links to Time Entity and Information Object. Note that each node in the graph represents a *pattern*, i.e., there may be more modeling details inside it.

field expeditions. The associated activity that leads to the acquisition of a physical sample is modeled through *Sampling Process*, which is just an event stub at the moment.

Information Object. This is a generic entity that is attached to any of the entities above and used to represent additional information pertaining to the entity but not directly part of the nature of the entity, e.g., website address, textual description, or additional identifiers.

13.3.2. Use of Dual Schema

Initial feedback from the data providers indicated that although the CPs sufficiently cover important notions in the domain that intuitively correspond to existing data, populating them is not as straightforward [9, Section 4.6]. The reason is because some of the CPs is an abstraction that may not directly align to concrete data model at a data provider's side. In addition, some data providers find reified relations, such as those modeled by the Agent Role pattern, difficult to populate because they are reluctant to maintain the URI of instances of that pattern.

In response to the challenges faced by data providers in publishing their data against the patterns, the project decided to introduce a layer of *simplified*

schema(s), which is called *producer-centric view*, between the patterns and the data sources. Producer-centric views are somewhat similar to the simpler schema part of the dual schema described in Chapter 10 [13], and the patterns act as the more complex schema. Furthermore, this new layer of schema can be provider-specific: different data providers may have different simplified schemas, although for now, the project defines only one “default” simplified schema, i.e., all data providers currently use the same producer-centric view.

Producer-centric views are implemented as a lightweight ontology that contains only vocabulary declarations (i.e., declaration class, property, and individual names) and for each property, domain and range restrictions. These domain and range restrictions are unscoped¹¹ and *not* intended to be merged into the patterns. These assertions are useful for data providers to guide them when materializing a property. This is different from the way we created the simpler schema component of the dual schema in Chapter 10 [13] in which the domain and range restrictions remain scoped. Still, the idea remains the same in that a producer-centric view encapsulates shortcuts inside a CP or spanning several CPs.

For example, the property `hasChiefScientist` is employed to connect a cruise to a person who acts its chief scientist; so the domain of this property is `Cruise` and the range is `Person`. For data providers, this indicates that when generating an RDF triple with `hasChiefScientist` as the predicate, the subject has to be an instance of `Cruise` and the object must be an instance of `Person`. Of course, OWL semantics does not preclude one to assert an RDF triple whose subject is not explicitly asserted to be an instance of `Cruise` (or the object is not explicitly asserted to be an instance of `Person`). But data providers in this step do not use the view to find inferences, but rather, to *populate* the patterns. The correspondence can then be made between the patterns (involving the `Cruise`, `Agent Role`, and `Person` patterns) and the view, which can be expressed as the following rule:

$$\begin{aligned} \text{Cruise}(x) \wedge \text{providesAgentRole}(x, y) \wedge \text{ChiefScientistRole}(y) \\ \wedge \text{isPerformedBy}(y, z) \wedge \text{Person}(z) \rightarrow \text{hasChiefScientist}(x, z) \end{aligned}$$

where `ChiefScientistRole` is a specialization of the `Agent Role` pattern. Note that, as we illustrated in Chapter 10, to populate the patterns we essentially follow the opposite direction of the arrow in the rule above.

The domain and range restrictions of a property in a producer-centric view may be expressed in the form of class union. This indicates options in populating the property. For example, the property `hasCreator` has `Dataset` \sqcup `DigitalObject` as the domain and `Organization` \sqcup `Person` as the range. This means that there can be at most four possible combinations of pairs that populate `hasCreator`. This is in a sense under-specified since one combination may not necessarily make sense from the perspective of the data providers. From usability perspective, however, data providers found that expressing the domain and range of a property this way makes the view readable and easy to understand.

The resulting architecture is depicted in Fig. 13.3 and now works as follows: data providers generate RDF triples for their data according to the vocabulary

¹¹See Chapter 1 [11] for the distinction between scoped and unscoped domain and range restrictions.

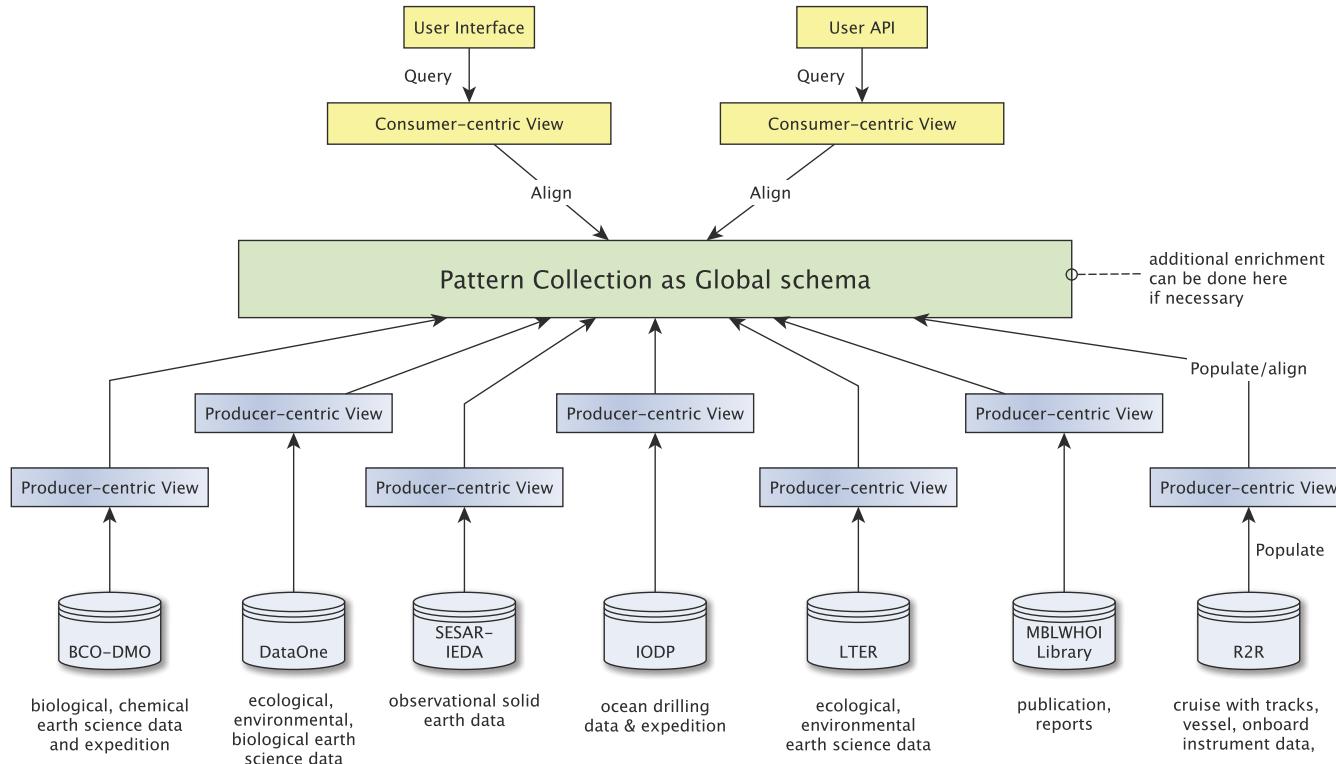


Figure 13.3. GeoLink architecture

defined in their producer-centric view and then, a set of transformation rules, which can be implemented as SPARQL CONSTRUCT queries, transform those triples into ones that conform the patterns. Now that the architecture allows a producer-centric view, it does make sense to extend the idea from the data consumer's perspective. That is, the architecture allows multiple *consumer-centric* views to be defined on top of the global schema. How a consumer-centric view is actually designed depends on the specific use cases of the front-ends. Furthermore, a consumer-centric view may or may not be the same as a producer-centric view. This keeps the architecture flexible and extensible enough to cover different types of users and different data providers.

One rather glaring issue with regards to linked data integration is the fact that different data sources may have overlapping content. In the GeoLink project, for example, data about some cruises may appear in both R2R and BCO-DMO repositories or data about the same person may appear in multiple repositories. Of course, since each repository is maintained independently from each other, the same person or the same cruise may be associated with different URIs. When data about this person or cruise is searched by a user, it is expected that the result does not only contain all relevant data for the person or cruise, but also is presented in a consolidated manner. That means, different URIs that refer to the same entity have to be directly connected by a mapping.¹² This mapping is asserted through triples where the subject and object refer to the same entity and the predicate is an appropriate linking predicate such as `owl:sameAs`, `skos:exactMatch`, or `skos:closeMatch`. Front-ends can then make use of these triples to customize a consolidated presentation of the data.

13.4. Concluding Remarks

We have presented our experience in using ODPs for the purpose of data integration in ocean science. The use of ODPs allows us to develop a flexible and extendible framework for ontology-based linked data integration. In particular, ODPs can be used to form a modular global schema, while the dual schema approach adds another degree of flexibility for data providers. Since the project is still ongoing, a more comprehensive evaluation to confirm these advantages of ODPs is still in the making. However, a somewhat preliminary, qualitative evaluation with the data providers can attest to the following feedback from data providers (see Krisnadhi [9] for more details).

- Data providers found that ODPs indeed offer flexibility since one can cherry-pick which pattern to align to, which is harder to do if monolithic domain ontologies are used.
- Abstractions in ODPs may introduce some gap to the conceptualization that each data provider has and this can indeed be bridged through the use of producer-centric views. This essentially confirms an advantage of using the dual schema approach that we espoused in Chapter 10 [13].

¹²The process of discovering these URIs is called *coreference resolution* for which the discussion is beyond the scope of this chapter.

- Data providers currently involved in the project are convinced that dual schema approach would help any new data provider who wishes to join the integration later after it has been established. This confirms the extendibility of the approach.

Acknowledgements. This work was supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*.

Bibliography

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.
- [2] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
- [3] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Elsevier Science, 2012.
- [4] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres. OWL: yet to arrive on the web of data? In C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, *WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.
- [5] E. J. Hackett et al. Ecology transformed: NCEAS and changing patterns of ecological research. In G. M. Olson et al., editors, *Science on the Internet*, pages 277–296. MIT Press, Cambridge, MA, 2008.
- [6] S. E. Hampton and J. N. Parker. Collaboration and productivity in scientific synthesis. *BioScience*, 61(11):900–910, 2011.
- [7] P. Jain, P. Hitzler, P. Z. Yeh, K. Verma, and A. P. Sheth. Linked data is merely more data. In D. Brickley, V. K. Chaudhri, H. Halpin, and D. McGuinness, editors, *Linked Data Meets Artificial Intelligence: Papers from the 2010 AAAI Spring Symposium, Technical Report SS-10-07, Stanford, California, USA, March 22-24, 2010*, pages 82–86. AAAI Press, 2010. Available at <http://www.aaai.org/Library/Symposia/Spring/ss10-07.php>.
- [8] K. Janowicz. Modeling Ontology Design Patterns with domain experts – a view from the trenches. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [9] A. Krisnadhi. *Ontology Pattern-Based Data Integration*. PhD thesis, Wright State University, 2015.
- [10] A. Krisnadhi. The role patterns. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.

- [11] A. Krisnadhi and P. Hitzler. Modeling with Ontology Design Patterns: Chess games as a worked example. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [12] A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. A. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. W. Finin, P. Ji, M. B. Jones, N. Karima, K. A. Lehnert, A. Mickle, T. W. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The GeoLink modular oceanography ontology. In M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 2015.
- [13] A. Krisnadhi, N. Karima, P. Hitzler, R. Amini, M. Cheatham, V. Rodríguez-Doncel, and K. Janowicz. Ontology Design Patterns for linked data publishing. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [14] A. A. Krisnadhi, Y. Hu, K. Janowicz, P. Hitzler, R. A. Arko, S. Carbotte, C. Chandler, M. Cheatham, D. Fils, T. Finin, P. Ji, M. B. Jones, N. Karima, K. A. Lehnert, A. Mickle, T. Narock, M. O'Brien, L. Raymond, A. Shepherd, M. Schildhauer, and P. Wiebe. The GeoLink framework for pattern-based linked data integration. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015.*, volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [15] M. Lenzerini. Data integration: A theoretical perspective. In L. Popa, S. Abiteboul, and P. G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002.
- [16] D. Oberle. *Semantic Management of Middleware*, volume 1 of *Semantic Web and Beyond: Computing for Human Experience*. Springer, 2006.
- [17] V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In Q. Li, S. Spaccapietra, E. S. K. Yu, and A. Olivé, editors, *Conceptual Modeling – ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings*, volume 5231 of *Lecture Notes in Computer Science*, pages 128–141. Springer, 2008.
- [18] V. Presutti, A. Gangemi, S. David, G. Aguado de Cea, M. C. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. A library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. Technical report, Deliverable D2.5.1, NeOn Project, 2008.
- [19] O. Reichman, M. B. Jones, and M. P. Schildhauer. Challenges and opportu-

- nities of open data in ecology. *Science*, 331(6018), 2011.
- [20] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, editors, *The Semantic Web – ISWC 2014 – 13th International Semantic Web Conference, Riva del Garda, Italy, October 19–23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2014.
 - [21] U.S. Department of Transportation – Federal Highway Administration. Data integration primer: Challenges to data integration. Online at <http://www.fhwa.dot.gov/asset/dataintegration/if10019/dip06.cfm>; Last accessed: May 18, 2016.
 - [22] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information-a survey of existing approaches. In *Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, 2001*, pages 108–117, 2001.
 - [23] J. You. Geoscientists aim to magnify specialized web searching. *Science*, 347(6217):11–11, 2015.

Chapter 14

Towards Enabling Traceability in Supply Chains via Ontology Design Patterns

Monika Solanki, University of Oxford

14.1. Introduction

One of the most important challenges in supply chains and logistics is information integration. Sharing of data and knowledge in a standardised manner along the supply chain is crucial not only to enable visibility, i.e., tracking and tracing of artifacts, but also to enable the more effective management of the supply chain. Business relations are increasingly globalised and loosely-coupled thus making both standards and technologies for information exchange essential. Sectors where this is particularly necessary are food and agriculture, and pharmaceuticals. This is due to the inherently great complexity of these supply chains, and the importance of tracking and tracing to comply with security, safety and regulatory requirements.

For information to be usefully shared, it must be interlinked and made available at all stages of the supply chain (with due regard for data ownership). Two critical issues that act as a hindrance to enabling information exchange in existing supply chain processes are the following:

- In supply chains, particularly in the agri-food sector, the flow of data is restricted by a very conservative “need-to-know” attitude such that essentially information flows only “one up, one down”. Aligned to this is the fact that existing systems deployed by individual partners in the supply chains do not support the sharing of information across multiple partners.
- Although a very large number of items are scanned in supply chains, and each actor records these events in their systems, there is no linkage of these data items across actors. This is due both to the cultural barriers to information exchange in the sector and the current set of technological solutions.

Barcodes and more recently RFID tags have provided initial solutions to this challenge. GS1¹, the standards organisation which manages barcodes, has provided various standards to facilitate end-to-end traceability and information sharing along the supply chain. The Electronic Product Code Information Services (EPCIS)² and the Core Business Vocabulary³, collectively provide specifications for the representation of product traceability information[5].

EPCIS provides a data model serialised as an XML schema, for capturing information artifacts that encapsulate the geographical progress and status of an item or set of items at each step of the supply chain. This is achieved by capturing data generated within the business context of scanning a barcode or RFID tag and encapsulating it within the abstraction of an “event”. A critical limitation of the current EPCIS specification is that though it does propose a mechanism to exchange and share data, the EPCIS XML schemas define only the structure of the data to be recorded. The semantics of data and data curation processes are informally defined. Their interpretation is left up to the individual EPCIS specification implementing engines, thereby greatly increasing the possibility of interoperability issues arising between supporting applications, e.g., validation and discovery Web services built over the event repositories.

To overcome the limitations and hindrances highlighted above, we propose the concept of event based “linked pedigree” [9, 12] – interlinked datasets described in RDF, curated by consuming real time EPCIS events in the supply chain and encapsulated as linked data, that enable the capture of a variety of tracking and tracing information such as the *Chain of Custody* (CoC) and *Chain of Ownership* (CoO) about products as they move among the various trading partners.

The notion of linked pedigrees is motivated by the widely prevalent use of pedigrees for tracking and tracing commodities in the pharmaceutical industry⁴. Pedigree or electronic pedigree (e-pedigree) is an audit trail that records the path and ownership of a drug as it moves through the supply chain, in which each stakeholder involved in the manufacture or distribution of the drug adds traceability information to the pedigree. As pedigrees are a collection of traceability information contributed by multiple parties, in order to achieve interoperability between pedigree handling systems, conformance to standards by all the parties involved is crucial. The modelling of pedigrees using GS1 specifications and representation using Semantic Web standards and linked data technologies provides a universally standardised mechanism to exchange traceability information.

Linked pedigrees specifically overcome a significant limitation prevalent in conventional pedigree exchange - that of information being available only from partners one-up or one-down in the supply chain. Data-driven supply chain systems, empowered by linked pedigrees can enable the sharing of information across multiple partners and bridge the social barriers highlighted above. Dereferencing URIs make it possible to sequentially traverse the chain of pedigrees exchanged between partners and retrieve traceability information, given that adequate authentication, authorisation and access control mechanisms are in place.

¹<http://www.gs1.org/>

²<http://www.gs1.org/gsmp/kc/epcglobal/epcis>

³<http://www.gs1.org/gsmp/kc/epcglobal/cbv>

⁴<http://www.axway.com/products-solutions/b2b/life-sciences-solutions/epedigree>

In this chapter we present “OntoPedigree”⁵, a content ontology design pattern for the data modelling of linked pedigrees, that can be specialised and extended to define domain specific or indeed product specific pedigree ontologies. OntoPedigree exploits the *EPCIS Event Model* (EEM)⁶ based on the EPCIS specification, that enables the sharing and semantic interpretation of event data. In particular the chapter makes the following scientific contributions:

- We formalise the notion of a track and trace artifact, “linked pedigrees” using the underlying standards for Semantic Web and principles of linked data to represent traceability-specific domain knowledge in supply chains.
- We illustrate how EPCIS governing supply chain events can be exploited to generate linked pedigrees using the OntoPedigree design pattern.

To augment the knowledge-based representation of linked pedigrees, we propose an implementation framework that can be utilised for their generation and querying. In order to exemplify the application of our contribution as OntoPedigree driven linked pedigrees, we further provide motivating examples from the perishable goods and pharmaceuticals supply chain sectors.

The chapter is structured as follows: Section 14.2 presents a brief background on the EPCIS specification and the EEM ontology utilised in the development of OntoPedigree. Section 14.3 presents the requirements and competency questions we considered for the design of the pattern. Section 14.4 presents the intent, conceptual entities and the graphical illustration of OntoPedigree. It also highlights the relationship of OntoPedigree to other patterns and ontologies and provides details of implementation support for OntoPedigree. Section 14.5 presents the OWL axiomatisation of the pattern. Section 14.6 presents scenarios from the agri-food and pharmaceutical supply chain where OntoPedigree has been applied. Section 14.7 discusses related work and Section 14.8 presents conclusions.

14.2. Background

The EPCIS specification is an EPCglobal standard that supports a detailed representation of the location and state of a product as it moves between organisational boundaries and provides for sharing this information between entities or partners, in a technology-supplier independent way. The standard is data carrier neutral and can be used to exchange data found from RFID tags, barcodes and other data carriers.

An Electronic Product Code (EPC)⁷ is a universal identifier that gives a unique, serialised identity to a specific physical object. In most instances, EPCs are encoded on barcodes or RFID tags which can be used to track all kinds of objects including: trade items, fixed assets, documents, or reusable transport items.

EPCIS identifiers for events, products and locations are currently represented using URNs. Various formats for URNs that can be used for identifying the EPCs

⁵<https://w3id.org/pedigree#>

⁶<http://purl.org/eem#>

⁷http://www.gs1.org/gsmp/kc/epcglobal/tds/tds_1_6-RatifiedStd-20110922.pdf

have been prescribed in the GS1 EPC Tag Data Standard⁸. A typical identifier for classes of products is GTIN (Global Trade Item Number)⁹. An individual serialized item on the other hand is identified using SGTIN (Serialized Global Trade Item Number), e.g., two cases of the same product will have the same GTIN, but individual products in the cases will have different SGTINs, e.g.,

`urn:epc:id:sgtin:0614141.112345.400`

Similar identifiers¹⁰ are prescribed for RFID read points and business locations using `sln`, “Serialized Global Location Number”.

The EPCIS specification defines two kinds of data: event data and master data. Event data arises in the course of carrying out business processes, it grows over time and is captured through the EPCIS capture interface and made available for query through the EPCIS query interfaces. Some examples of event data that can be represented using the EPCIS specification are illustrated below:

- *At time T, the association of the following case tags to the following pallet tag was created at palletizer #3, to fulfill order #1234.*
- *Between the time the case crossed the first beam and the second beam at location L, the following tag was read.*
- *At Time T, Object X was observed at Location L.*

Master data is additional data that provides the necessary context for interpreting the event data. An example of master data is *Location L refers to the distribution centre located at 123 Elm Street, Anytown, US..*

The EPCIS standard defines a generic event and four different physical event types, arising from supply chain activity across a wide variety of industries.

- *EPCISEvent* represents the generic EPCIS event.
- *ObjectEvent* represents an event that occurred as a result of some action on one or more entities denoted by EPCs.
- *AggregationEvent* represents an event that happened to one or more EPC-denoted entities that are physically aggregated (constrained to be in the same place at the same time, as when crates are aggregated to a pallet).
- *QuantityEvent*¹¹ represents an event concerned with a specific number of objects all having the same GTIN, but where the individual instances are not identified. For example a quantity event could report that an event happened to 200 boxes of widgets, without identifying specifically which boxes were involved.
- *TransactionEvent* represents an event in which one or more entities denoted by EPCs become associated or disassociated with one or more identified business transactions.

Each EPCIS event, recorded and registered against RFID tagged or barcoded artifacts has four information dimensions. It encapsulates the “what”, “when”, “where” and “why” of these artifacts at the barcode or RFID scan point.

⁸http://www.gs1.org/gsmp/kc/epcglobal/tds/tds_1_6-RatifiedStd-20110922.pdf

⁹<http://www.gs1.org/barcodes/technical/idkeys/gtin>

¹⁰We do not include details of these identifiers. The interested reader is referred to the Tag Data standard

¹¹Deprecated in EPCIS 1.1

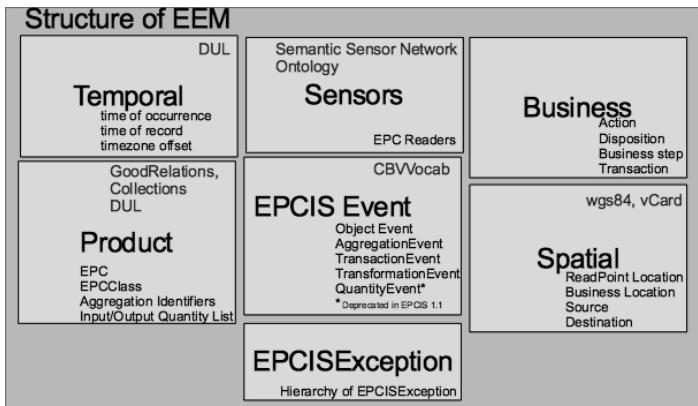


Figure 14.1. Structure of EEM and its alignment with external ontologies (noted in blue coloured text)

EEM is an OWL 2 DL ontology for modelling EPCIS events. EEM conceptualises various primitives of an EPCIS event that need to be asserted for the purposes of traceability in supply chains. Development of EEM was informed by a thorough review of the EPCIS specification and extensive discussions with trading partners implementing the specification. The modelling decisions[10] behind the conceptual entities in EEM highlight the EPCIS abstractions included in the ontology. In [12] a mapping between EEM and PROV-O¹², the vocabulary for representing provenance of Web resources has been defined. The event history can be interrogated using PROV-O for recovering provenance information associated with the events.

The EEM ontology structure and its alignment with various external ontologies is illustrated in Figure 14.1. The ontology is composed of modules that define various perspectives on EPCIS. The *Temporal* module captures timing properties associated with an EPCIS event. It is aligned with temporal properties in DOLCE+DnS Ultralite (DUL)¹³. Entities defining the EPC, aggregation of EPCs and quantity lists for transformation events are part of the *Product* module. The GoodRelations¹⁴ ontology is exploited here for capturing concepts such as an Individual Product or a lot (collection) of items, SomeItems of a single type. Information about the business context associated with an EPCIS event is encoded using the entities and relationships defined in the *Business* module. RFID readers and sensors are defined in the *Sensor* module. The definitions here are aligned with the SSN¹⁵ ontology. The *EPCISEException* module incorporates the hierarchy of the most commonly observed exceptions[14] occurring in EPCIS governing supply chains.

For further details on EEM and its applications in real world scenarios, the interested reader is referred to [2, 9, 10, 12, 13].

¹²<http://www.w3.org/ns/prov-o>

¹³<http://ontologydesignpatterns.org/ont/dul/DUL.owl>

¹⁴<http://purl.org/goodrelations/v1>

¹⁵<http://purl.oclc.org/NET/ssnx/ssn>

14.3. Requirements analysis

Pedigrees have been defined initially in EPCGlobal's ratified Pedigree Standard¹⁶ as follows: "A pedigree is a *certified* record that contains information about each distribution of a prescription drug. It records the sale of an item by a pharmaceutical manufacturer, any acquisitions and sales by wholesalers or repackagers, and final sale to a pharmacy or other entity administering or dispensing the drug. The pedigree contains *product information, transaction information, distributor information, recipient information, and signatures*."

While the above description of a pedigree is given within the context of pharmaceutical supply chains, the interpretation of the definition highlights certain key requirements for the design of a generic content ontology design pattern for pedigrees, that could be reused across multiple sectors.

Recently the concept of "Event based Pedigree"¹⁷ has been proposed that utilises EPCglobal's EPCIS specification for capturing events in the supply chain and generating traceability datasets based on a relevant subset of the captured events. OntoPedigree, the design pattern for the generation of linked pedigrees as presented in this paper, builds on the event based pedigree approach.

Below we outline some of the central requirements for pedigrees which form the basis of the entities defined in OntoPedigree.

- **Certification and digital signatures:** Pedigrees generated by supply chain stakeholders would need to be certified and digitally signed before they can be forwarded to other partners in the chain. Certification attributes include provenance information such as the organisational details associated with creating the pedigree and the Web service that generated it. Digital signatures would include the signature value and information needed to obtain the key associated with the signature.
- **Product information:** Master data about the products that are involved in the CoC/CoO must be interlinked or associated with the core pedigree information.
- **Location information:** Location master data such as the business name and address details associated with the source and destination organisations, identified using for e.g., their GLN (Global Location Number) must be interlinked.
- **Consignment information:** Pedigrees must include detailed information about the consignments being shipped or received such as identities of the pallets, cases, bundles and items, timing attributes for the various business steps undertaken at the stakeholder's facilities and the supply chain context associated with the shipment, i.e., pedigrees need to indicate their status in terms of the point (initial, intermediate or final) in the supply chain where they were generated.
- **Transaction information:** Transactional information such as identifiers of the purchase orders being fulfilled by the shipment are required to be incorporated in the pedigrees.

¹⁶Version 1.0, 2007, <http://www.gs1.org/gsmp/kc/epcglobal/pedigree>

¹⁷http://www.gs1.org/docs/healthcare/Healthcare_Traceability_Pedigree_Background.pdf

- **Partner pedigree information:** Apart from the pedigree initiated and created by the first partner in the supply chain, all other pedigrees must include information about the pedigrees for the stakeholders in the immediate upstream or downstream of the supply chain.

As highlighted in Section 14.1, a linked pedigree is a dataset, represented as a named graph[4], identified using an http URI, described and accessed using linked data principles and represented using the RDF data model. The definition of a pedigree must include URIs for certification, product, transaction and consignment information asserted in the trading partner’s knowledge base. It must include URIs to the pedigrees sent by the immediate upstream or downstream partners.

14.3.1. Competency questions

Given the requirements above, we define certain competency questions that govern the design of an ontology pattern for pedigrees.

- Who is the creator of the pedigree ?
- What is the supply chain creation status of a given pedigree?
- Which are the business transactions recorded against a particular consignment?
- What are the events associated with pedigrees created between dates X and Y?
- Which products have been shipped together?
- Which other pedigrees are included in the received pedigree?

14.4. Pattern description and Graphical representation

14.4.1. Intent

OntoPedigree provides a minimalistic abstraction and defines conceptual entities for the modelling of semantically enriched knowledge required to enhance data visibility in a supply chain. The pattern can be specialised to define domain specific pedigrees.

14.4.2. Conceptual Entities

Some of the key concepts and relationships encapsulating the data model defined by the pattern are described below. Note that several entities in OntoPedigree extend from or exploit concepts and relationships defined in PROV-O and EEM as illustrated in Figure 14.2.

- **Pedigree:** A concept defining the notion of a pedigree as an entity capturing certified information for a stakeholder in the supply chain.
- **PedigreeCreationStatus:** An entity defining the status of the creation a pedigree. Valid instances include `InitialPedigree`, `IntermediatePedigree` or `FinalPedigree`.

- **PedigreeCreator:** An entity linking the pedigree to its creating organisation.
- **PedigreeCreationService:** An entity representing the Web service used for the generation of the pedigree.
- **pedigreeCreationTime:** A datatype property linking the pedigree to the time of its creation.
- **hasSerialNumber:** An entity linking the pedigree to its uniquely assigned serial number.
- **hasTransactionInfo:** An entity defining the relationship between the pedigree and the conceptual entity capturing transaction information related to the sale, movement, return or transfer of goods.
- **hasProductInfo:** An entity defining the relationship between the pedigree and the conceptual entity capturing product information.
- **hasConsignmentInfo:** An entity defining the relationship between the pedigree and the conceptual entity capturing consignment information.
- **hasReceivedPedigree:** An entity defining the relationship between the pedigree and any receiving pedigrees from stakeholders in the upstream or downstream of one or more participating supply chains. Examples of such receiving pedigrees include,
 - A pedigree defined by a repacking trader, who combines goods from several consignments received from upstream stakeholders.
 - A pedigree defined by a downstream retailer who returns goods to an upstream trader.

14.4.3. Reusing related patterns and ontologies in OntoPedigree

Traceability information in OntoPedigree, regarding consignments and transactions are specified as sets of EPCIS events described using EEM [10] ontology. Certification information is defined using the PROV-O vocabulary. Extending from PROV-O serves the dual purpose of being able to attach provenance to pedigrees as well as facilitates the querying of pedigrees for certified knowledge using a dedicated vocabulary. Product master data is specified using domain specific vocabularies that in many scenarios exploit the GoodRelations ontology¹⁸.

Pedigrees generated using OntoPedigree are based on GS1 standards. Supply chain processes that implement these standards for the generation of linked pedigrees can declare their conformance and enforcement to these standards using the Standards Enforcer content ontology design pattern Pattern(SEP)¹⁹ and link it with the pedigrees. The added value of this interlinking is that besides the certification information included in the pedigree, it serves as an additional validation when the pedigree is received and dereferenced by other partners.

¹⁸<http://purl.org/goodrelations/v1>

¹⁹<http://windermere.aston.ac.uk/~monika/ontologies/Standards-Enforcer-Pattern.owl>, http://ontologydesignpatterns.org/wiki/Submissions:Standard_Enforcer_Pattern

14.4.4. Graphical representation

Figure 14.2 illustrates the graphical representation of OntoPedigree. It depicts the entities defined for the pattern and their relationships with entities from PROV-O, EEM and GoodRelations.

14.4.5. Implementation support for OntoPedigree

In order to facilitate the creation of linked pedigrees that exploits EEM and OntoPedigree, we have implemented the LinkedEPCIS reference implementation and Java library²⁰. An important component of the library is the “LinkedPedigreeGenerator” which automates the generation of linked pedigrees by querying the triple store for event URIs. Additionally, URIs for the product master data, made available as linked data are retrieved. Consignment information is populated with spatial data for locations from DBpedia and Geonames. The pedigree status is dynamically set, based on the point at which the pedigree is generated. LinkedEPCIS thus provides the implementation support required for the uptake of OntoPedigree by supply chain stakeholders, to be exploited in their own IT infrastructure.

14.5. OntoPedigree Axiomatisation

A pedigree is required to have certain mandatory properties and relationships as part of its definition. In this section we present an incremental axiomatisation of OntoPedigree(**ped**) using the OWL Manchester syntax. As noted earlier, several entities in OntoPedigree extend or exploit the definitions in PROV(**prov**) and EEM(**eem**).

- A pedigree is a PROV entity

```
Class: ped:Pedigree
SubClassOf:
  prov:Entity
```

- A pedigree must include a creation time, a serial number and describe its status.

```
Class: ped:Pedigree
SubClassOf:
  (hasPedigreeStatus exactly 1
    ped:PedigreeStatus)
  and (hasSerialNumber exactly 1
    rdfs:Literal)
  and (pedigreeCreationTime exactly 1
    xsd:DateTime)
```

- It must assert the creating organisation and may include the service used for its creation.

```
Class: ped:Pedigree
SubClassOf:
  (prov:wasAttributedTo exactly 1
    ped:PedigreeCreator),
  (prov:wasGeneratedBy only
    ped:PedigreeCreationService)
```

²⁰<http://github.com/nimonika/LinkedEpcis/>

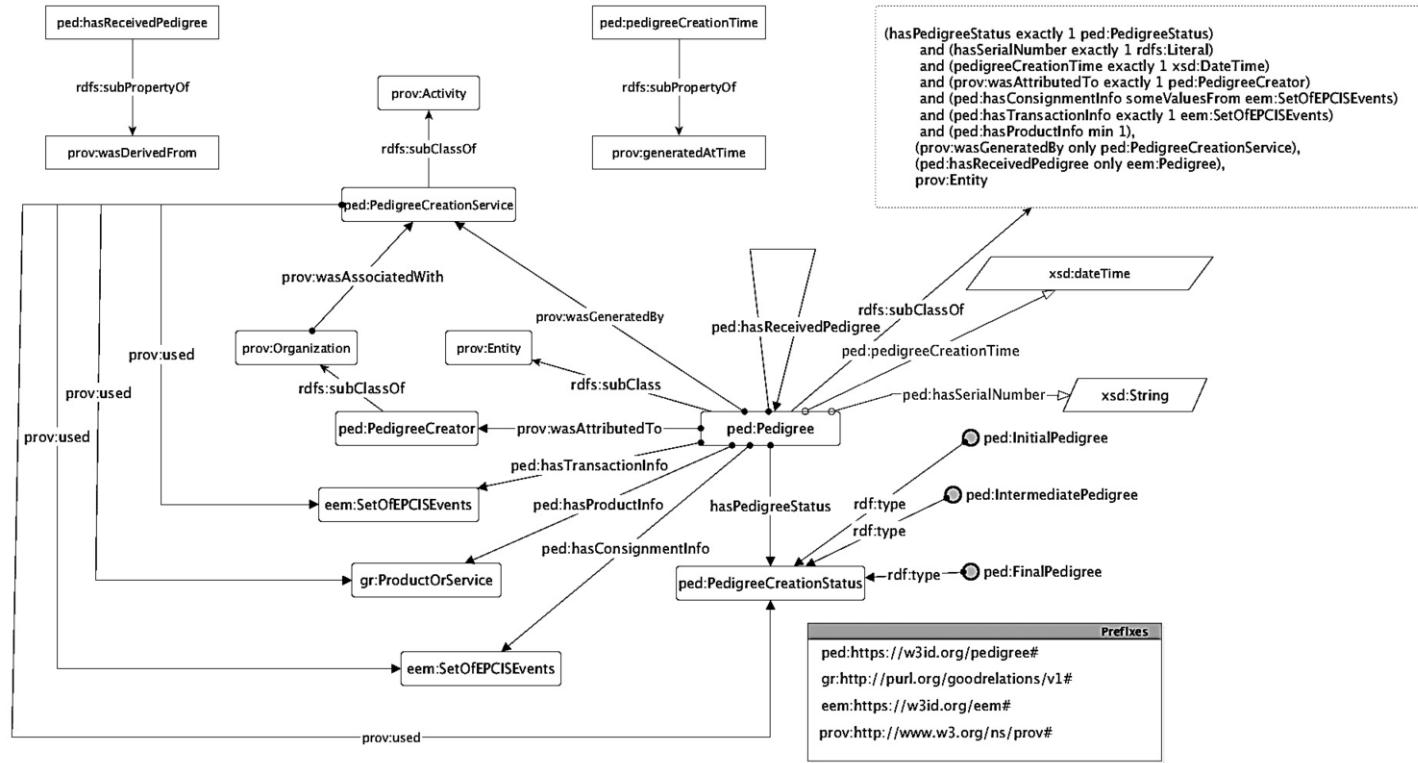


Figure 14.2. Graphical Representation of OntoPedigree

```

Class: ped:Pedigree
SubClassOf:
  (hasPedigreeStatus exactly 1 ped:PedigreeStatus)
  and (hasSerialNumber exactly 1 rdfs:Literal)
  and (pedigreeCreationTime exactly 1 xsd:DateTime)
  and (prov:wasAttributedTo exactly 1 ped:PedigreeCreator)
  and (ped:hasConsignmentInfo someValuesFrom eem:SetOfEPCISEvents)
  and (ped:hasTransactionInfo exactly 1 eem:SetOfEPCISEvents)
  and (ped:hasProductInfo min 1),
  (prov:wasGeneratedBy only ped:PedigreeCreationService),
  (ped:hasReceivedPedigree only eem:Pedigree),
  prov:Entity

```

Figure 14.3. Manchester syntax serialisation of OntoPedigree

- It must include at least one consignment relation, a transaction relation and at least one relationship capturing product master data.

```

Class: ped:Pedigree
SubClassOf:
  (ped:hasConsignmentInfo someValues
    eem:SetOfEPCISEvents)
  and (ped:hasTransactionInfo exactly 1
    eem:SetOfEPCISEvents)
  and (ped:hasProductInfo min 1)

```

- Finally, it may include links to pedigrees received from the immediate upstream or downstream partners.

```

Class: ped:Pedigree
SubClassOf:
  (ped:hasReceivedPedigree only eem:Pedigree)

```

Combining the above assertions, the overall definition of **Pedigree** is illustrated in Figure 14.3.

14.6. Exemplifying OntoPedigree

In this section we present scenarios from the perishable goods and pharmaceutical supply chains that exemplify the use of OntoPedigree in the exchange of traceability information. Perishable goods is one of the scenarios considered in the FI-PPP projects SmartAgrifood²¹ and FIspace²².

14.6.1. Perishable Goods supply chains

The lifecycle of perishable goods e.g., tomatoes in the agri-food sector, is a complex process until they reach the end consumer because of the number of involved stakeholders and the diverse set of data that is produced. The tomato

²¹<http://www.smartagrifood.eu/>

²²<http://www.fispace.eu/>

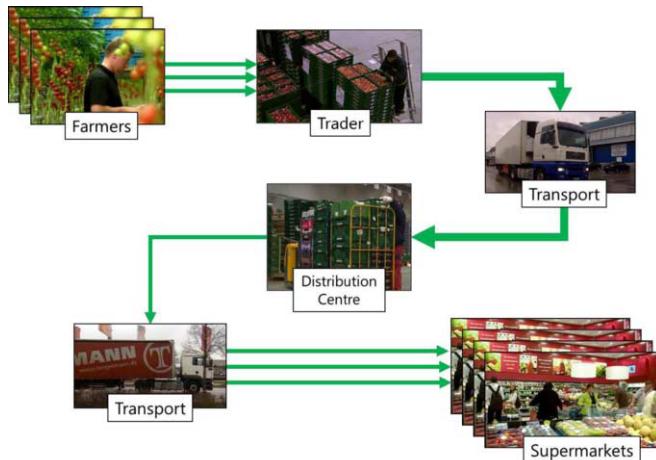


Figure 14.4. Generalised agri-food chain scenario for tomatoes

supply chain involves thousands of farmers, hundreds of traders and few retail groups. Figure 14.4 shows a generalised food chain scenario with a reduced level of complexity. This scenario covers 90% of the supply scenarios for fresh food products. The general workflow involving the capturing of events, generation of linked pedigrees and exchange of pedigrees related to the sale of tomatoes between stakeholders such as the farmer (Franz), the trader (Joe), the distribution centre (FreshFoods Inc) and the supermarket (Orchard) is outlined below. We assume that all supply chain trading partners have a supporting EPCIS implementation infrastructure installed that can be exploited for capturing and querying event and pedigree datasets.

- Franz farmer specialises in growing tomatoes. The packaging of tomatoes is done in punnets, each of which are tagged with RFID labels. Shipment of tomatoes to downstream partners is done in cardboard boxes each of which is tagged with a RFID tag.
- Joe trader bundles tomatoes procured from multiple farmers to larger product batches before dispatching them to distribution centres.
- Freshfoods Inc. sources tomatoes from multiple traders and splits up large product batches into smaller batches for distribution to retail supermarkets.
- Orchards is a supermarket that receives fresh produce from distribution centres such as Freshfoods Inc.

Joe trader requests pedigree information on an identified tomato batch that has been delivered to him by Franz farmer. The request is made by RESTfully invoking Franz farmer's agent which is part of the FMS (farm management system) installed at Franz farmer's end [1]. Joe trader receives an authenticated and certified message containing the pedigree URI, where the pedigree has been generated using the OntoPedigree design pattern. Joe trader's agent dereferences the URI and receives the pedigree dataset. Object property value resources in the pedigrees are asserted using EPCIS event data URIs.

```

fsc:FranzTomatoFarmerPedigree123 rdf:type ped:Pedigree;
prov:wasAttributedTo <http://www.joetomatotrader/foaf>;
ped:pedigreeCreationTime "2014-06-05T07:53:14.189+01:00"
                           ^^xsd:dateTime;
ped:hasSerialNumber "tomPed123"^^xsd:String;
ped:hasStatus ped:Initial;
ped:hasConsignmentInfo fci:FranzFarmerObjectEvent10,
                        fci:FranzFarmerAggregationEvent6;
ped:hasTransactionInfo fti:FranzFarmerShippingEvent12;
ped:hasProductInfo ftp:FranzTomatoesMay2013Data.

jsc:JoeTomatoTraderPedigree456 rdf:type ped:Pedigree;
prov:wasAttributedTo <http://www.joetomatotrader/foaf>;
ped:pedigreeCreationTime "2014-06-08T03:67:09.175+01:00"
                           ^^xsd:dateTime;
ped:hasSerialNumber "joeTradePed456"^^xsd:String;
ped:hasStatus ped:Intermediate;
ped:hasConsignmentInfo jci:JoeTraderObjectEvent20,
                        jci:JoeTraderObjectEvent30;
ped:hasTransactionInfo jti:JoeTraderTransactionEvent40;
ped:hasProductInfo jpi:JoeTradesMay2013Info.
ped:hasReceivedPedigree fsc:FranzTomatoFarmerPedigree123,
                         bsc:BobTomatoFarmerPedigree123.

```

Figure 14.5. Linked pedigrees created using OntoPedigree and exchanged between the supply chain partners

Joe trader combines the tomato produce received from Franz with those received from other farmers (e.g., Bob) into shipments which are then forwarded to Freshfood Inc. On receiving a pedigree request from Freshfood Inc, Joe trader's agent sends the pedigree which includes URIs to the pedigree provided by Franz farmer and Bob farmer.

The pedigrees generated and exchanged in the workflow defined above are illustrated in Figure 14.5.

The significant advantage of exchanging traceability information using linked pedigrees over conventional mechanisms is that the pedigree received by FreshFood Inc. from Joe trader includes URIs to pedigree datasets provided by Franz farmer and Bob farmer, even though they are not FreshFood Inc's one-up or one-down partners. Consuming EPCIS event data curated as linked data to generate and exchange linked pedigrees as outlined above can help derive implicit knowledge that can expose inefficiencies such as shipment delay, inventory shrinkage and out-of-stock situations.

14.6.2. Pharmaceutical Supply chains

We outline the scenario of a pharmaceutical supply chain, where trading partners exchange product track and trace data using linked pedigrees. Figure 14.6

illustrates the flow of data for four of the key partners in the chain.

The *Manufacturer* commissions²³, i.e., assigns an EPC(Electronic Product Code) to the items, cases and pallets. The items are packed in cases, cases are loaded onto pallets and pallets are shipped. At the *Warehouse*, the pallets are received and the cases are unloaded. The cases are then shipped to the various *Distribution centers*. From the Distribution centers the cases are sent to retail *Dispenser* outlets, where they are received and unpacked. Finally, the items are stacked on shelves for dispensing, thereby reaching their end-of-life in the product lifecycle.

As the serialised items, cases and pallets move through the various phases of the supply chain at a trading partner's premises, EPCIS events are generated and recorded at several RFID reader nodes.

When the pallets with the cases are shipped from the manufacturer's premises to the warehouse, pedigrees encapsulating the minimum set of EPCIS events are published at an URI based on a predefined URI scheme. At the warehouse, when the shipment is received, the URI of the pedigree is dereferenced to retrieve the manufacturer's pedigree. When the warehouse ships the cases to the distribution center, it incorporates the URI of the manufacturer's pedigree in its own pedigree definition. As the product moves, pedigrees are generated with receiving pedigrees being dereferenced and incorporated, till the product reaches its end-of-life stage.

14.7. Related work

While supply chain information visibility[8] has received significant attention in recent years, to the best of our knowledge, our work is the first attempt in utilising Semantic Web standards and linked data principles for the representation of EPCIS events and for exploiting the representation for provenance-based tracking and tracing.

Closely related to our work is the use of Semantic Web technologies for capturing and managing data across the supply chain as first proposed in [3] although the focus was on the environmental impact of food in the organic food supply chain. The CASSANDRA project ²⁴ proposes the concept of a virtual *data pipeline* that connects entities and gathers and distributes data according to predefined conditions. However there is no provenance associated with the datasets being gathered and distributed. In [7] the authors present a contextual architecture for making supply chain data available to applications designed for customs authorities. A crucial limitation of this approach is the use of a centralised linked data store for crawled linked open data.

In [6] the authors present a solution that utilises both RFID and GPS for tracking and tracing of international shipments, although the solution uses EPCIS, the focus there is on the implementation of a system, rather than utilising a traceability artifact such as a pedigree as done in our approach.

In [11] a data model and algorithm for managing and querying event data has been proposed. . The data model is illustrated as an extended entity relationship

²³associating the serial number with the physical product

²⁴<http://www.cassandra-project.eu/>

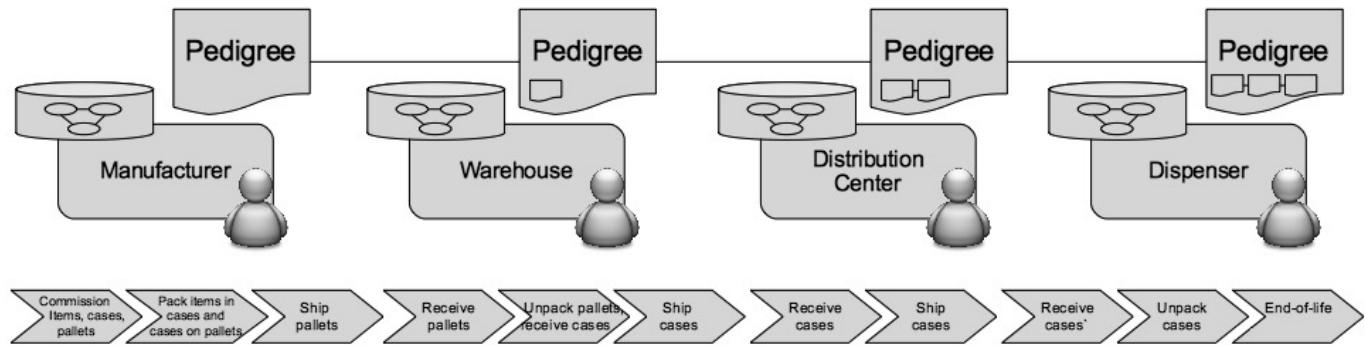


Figure 14.6. Trading partners in a pharmaceutical supply chain and the flow of information

diagram and is close in spirit to EEM as proposed in this paper. A critical limitation of this model is that it is overlayed on top of relational databases and is not available in a form that can be shared, reused between organisations as linked data.

In [15] the authors propose to use the InterDataNet (IDN)²⁵ framework for applying a RESTful architecture and linked data principles to facilitate the sharing of EPCIS data. The proposed approach suffers from several critical limitations. No reusable and shared data model is provided that can serve as the metadata layer for representing and reasoning over EPC-data. The encapsulation of information as IDN documents imposes an additional layer which may significantly affect performance of querying applications.

The Fosstrack²⁶ EPCIS modules provide a set of implementations for the EPCIS 1.0.1 specification, whereas our LinkedEPCIS library provides an API for the latest EPCIS 1.1 specification. Another limitation of the existing Fosstrack library is that any attributes added as an extension element to the existing schema cannot be directly captured and queried for through the interfaces provided in the API. This is however straightforward in LinkedEPCIS.

14.8. Conclusions

Data visibility in supply chains has received considerable attention in recent years due to existing siloed information management architectures and prevalent social barriers regarding sharing of data among partners. In this chapter we have proposed linked pedigrees, that enable the interlinking of diverse information across supply chains, thereby providing a solution to the hindrances outlined in Section 14.1.

We have defined a design pattern “OntoPedigree” that provides a minimalistic abstraction for designing domain specific pedigree ontologies. OntoPedigree builds upon EPCIS, a GS1 standard for facilitating event based traceability and PROV-O, a vocabulary for specifying provenance of linked datasets. OntoPedigree can be mapped with other content ontology patterns such as SEP to declare the conformance of supply chain processes to these standards when they generate pedigrees using OntoPedigree.

The pattern design is supported by a robust set of requirements and a rigorous set of competency questions. An OWL axiomatisation of OntoPedigree has been provided. Finally, an implementation framework that facilitates the seamless uptake of the generation of linked pedigrees in supply chains has been presented. It is worth noting that OntoPedigree is domain independent and can be widely applied to most scenarios of traceability. We strongly believe that linked pedigrees generated using OntoPedigree can make a significant difference to current visibility approaches in supply chains.

²⁵<http://www.interdatanet.org/>

²⁶<https://code.google.com/p/fosstrak/wiki/EpcisFeatures>

Bibliography

- [1] Alexandros Kaloxyllos et al. Farm management systems and the Future Internet era. *Computers and Electronics in Agriculture*, 89(0):130 – 144, 2012.
- [2] P. Bhattacharjee, M. Solanki, R. Bhattacharyya, I. Ehrenberg, and S. Sarma. Vacseen: A linked data-based information architecture to track vaccines using barcode scan authentication. In J. Malone, R. Stevens, K. Forsberg, and A. Splendiani, editors, *SWAT4LS*, volume 1546 of *CEUR Workshop Proceedings*, pages 39–48. CEUR-WS.org, 2015.
- [3] C. Brewster, H. Glaser, and B. Haughton. AKTive Food: Semantic Web based knowledge conduits for the Organic Food Industry. In *Proceedings of the ISWC Workshop Semantic Web Case Studies and Best Practice for eBusiness (SWCASE 05) , 4th International Semantic Web Conference, 7 November*, Galway, Ireland, 2005.
- [4] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05. ACM, 2005.
- [5] K. Främling, S. Parmar, V. Hinkka, J. Tätilä, and D. Rodgers. Assessment of epcis standard for interoperable tracking in the supply chain. In T. Borangiu, A. Thomas, and D. Trentesaux, editors, *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics*, volume 472 of *Studies in Computational Intelligence*, pages 119–134. Springer Berlin Heidelberg, 2013.
- [6] W. He, E. Tan, E. Lee, and T. Li. A solution for integrated track and trace in supply chain based on RFID and GPS. In *IEEE Conference on Emerging Technologies Factory Automation, 2009. ETFA 2009.*, 2009.
- [7] W. Hofman. Supply Chain Visibility with Linked Open Data for Supply Chain Risk Analysis. In *Proceedings of the 1st Workshop on IT Innovations Enabling Seamless and Secure Supply Chains (WITNESS2011)*, volume 769. CEUR Workshop proceedings, 2011.
- [8] Joris Hulstijn and Sietse Overbeek and Huib Aldewereld and Rob Christiaanse. Integrity of Supply Chain Visibility: Linking Information to the Physical World. In *CAiSE Workshops*, pages 351–365, 2012.
- [9] Monika Solanki and Christopher Brewster. Consuming Linked data in Supply Chains: Enabling data visibility via Linked Pedigrees. In *Fourth International Workshop on Consuming Linked Data (COLD2013) at ISWC*, volume Vol-1034. CEUR-WS.org proceedings, 2013.
- [10] Monika Solanki and Christopher Brewster. Representing Supply Chain Events on the Web of Data. In *Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE) at ISWC*. CEUR-WS.org proceedings, 2013.
- [11] Nguyen, Tuyen and Lee, Young-Koo and Jeong, Byeong-Soo and Lee, Sungyoun. Event Query Processing in EPC Information Services. In *Proceedings of the 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, SITIS '07, pages 159–166, Washington, DC, USA, 2007. IEEE Computer Society.
- [12] M. Solanki and C. Brewster. EPCIS event based traceability in pharmaceuti-

- cal supply chains via automated generation of linked pedigrees. In Peter Mika et al., editor, *Proceedings of the 13th International Semantic Web Conference (ISWC)*. Springer-Verlag, 2014.
- [13] M. Solanki and C. Brewster. Modelling and Linking transformations in EPCIS governing supply chain business processes. In Hepp, Martin; Hoffner, Yigal (Eds.), editor, *Proceedings of the 15th International Conference on Electronic Commerce and Web Technologies (EC-Web 2014)*. Springer LNBIP, 2014.
 - [14] M. Solanki and C. Brewster. Monitoring EPCIS Exceptions in linked traceability streams across supply chain business processes. In *Proceedings of the 10th International Conference on Semantic Systems (SEMANTiCS)*. ACM-ICPS, 2014.
 - [15] Turchi, S. and Ciofi, L. and Paganelli, F. and Pirri, F. and Giuli, D. Designing EPCIS through Linked Data and REST principles. In *20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2012*, pages 1–6, 2012.

Part III

Selected Examples of Ontology Design Patterns

This page intentionally left blank

Chapter 15

The Information Realization Pattern

Aldo Gangemi, Paris Nord University, France and ISTC-CNR, Rome, Italy

Silvio Peroni, DASPLab, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

15.1. Introduction

This chapter presents some ontology design patterns to formally represent Information Entities (IE). IE include e.g. words, sentences, documents, movies, pictures, paintings, performances, software programs, concepts, etc.

IE are at a core of language, social communication, knowledge, creativity, and norms. The amount of use cases involving IE is therefore huge. Semantics itself depends on IE, as well as copyright, artistic authenticity, reproducibility, software, legal validity, etc.

The patterns that we present here have applications in managing linguistic knowledge, document engineering, and creative works, but are useful in any use case that includes IE as first-order entities, i.e., not just as labels or comments. This distinction goes back to the Middle Age, and separates *de re* (“about the thing”) and *de dicto* (“about the name”) usages of IE. For example, we can talk about the city Rome (*de re* usage) – whose labels can be *Roma*, *Rome*, *Rom* ... – or about the name of Rome (*de dicto* usage) – e.g., when the name *Roma* is etymologically associated with *Rumon*, an old name of the river Tiber that means “to flow”.

Associated with, but more complex than the *de re* vs. *de dicto* distinction, there lie distinctions between a document (e.g., Dante’s Comedy or Coltrane’s Giant Steps music sheet), and what is expressed by a document (a fictional story, a musical work), between a content and its physical realization (e.g., Dante’s Comedy and a copy of the 1472 Numeister first printed edition of the Comedy), between an original IE and a copy (e.g., Redon’s Mystical Boat painting, and a fake of it vs. a poster reproducing it), or between an IE and its execution (e.g., a performance of Giant Steps, or the execution of a piece of software).

In each of those cases, the relation between an IE and “something else” depends on underlying social and cognitive practices (or “conceptual frames”, cf. chapter [10] in this book) that hint at the powerful role played by IE in human societies.

The chapter firstly introduces the *Semiotics* pattern, followed by *Information Realization*, as well as other patterns that are contained in public ontologies such as FRBR [16], SPAR [19], Dublin Core [2], CIDOC [13], MARK21 [15], etc.

15.2. A semiotic pattern

15.2.1. The pattern

The *Semiotics* ontology pattern¹ is based on the *Linguistic Meta-Model (LMM)* [20], and is, e.g., reused by the Ontolex-Lemon [11] in its core model, and applied (jointly with Earmark [17] and NIF [12] vocabularies) in the FRED [21] Open Knowledge Extraction tool.

It is shown in the Graffoo² diagram in Fig. 15.1.

The pattern *Semiotics* implements the basic ideas of semiotics:

- *References*: any individual, set of individuals, or situations from the world we are describing. They can have interpretations (meanings) and can be denoted by information entities. For example: *Bob, the set of Bob's relatives, or the fact that Bob is a professor*;
- *Meanings*: any (meta-level) object that explains something, or is intended by something, such as linguistic definitions, topic descriptions, lexical entries, thesaurus concepts, logical concepts or relations, etc. They can be “interpretants” for information entities, and “conceptualizations” for individuals and facts. For example, concepts such as *person, paragraph, having a role*;
- *Information entities*: any symbol that **expresses** a meaning, and **denotes** one or more references. They can be natural language terms, sentences or texts, symbols in formal languages, icons, or whatever can be used as a vehicle for communication – for example: the string “Bob”, the markup elements *p, agent, noun* and *verb* in XML documents. They have at least one meaning and can denote references. Moreover, each information entity can be an *expression* (e.g., the string “Bob”, defined by the class *1a:Expression*) realized in one or more *manifestations* (e.g., the string “Bob” contained in a particular XML file stored on somebody’s hard drive, defined by the class *1a:Manifestation*) having the same interpretation, which basically maps to a type-token distinction.
- *Linguistic acts*: any communicative situation including information entities, agents, meanings, references, and a possible spatio-temporal context

¹<http://www.ontologydesignpatterns.org/cp/owl/semiotics.owl>

²Graffoo, a *Graphical Framework for OWL Ontologies*, is an open source tool that can be used to present the classes, properties and restrictions within OWL ontologies, or subsections of them, as clear and easy-to-understand diagrams. The full specification is available at <http://www.essepuntato.it/graffoo> [7]

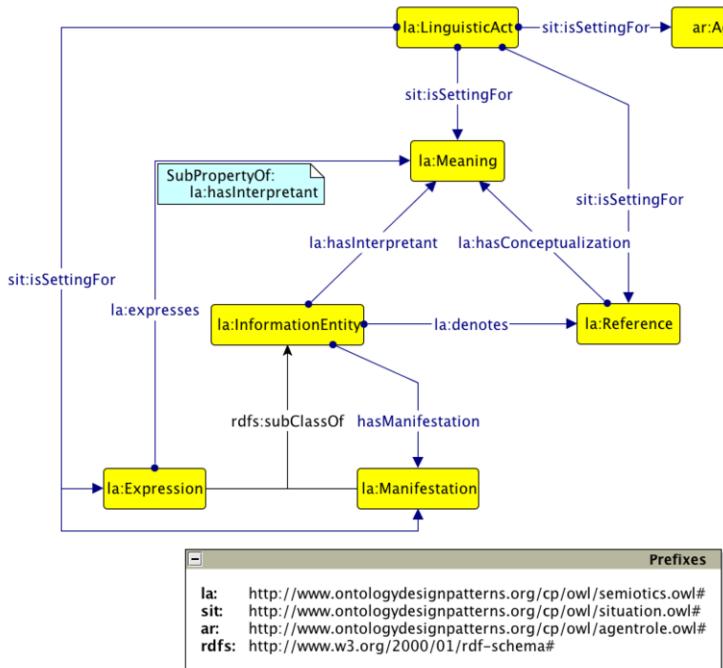


Figure 15.1. A diagram summarizing the ontology pattern *linguistic act*.

(i.e., when and/or where the act has been performed). For example, dialogues, tagging actions, writings.

- *Situations*: any state of affairs (or a role played in a state of affairs), which is referenced (is function) of a linguistic act, as previously exemplified.

For instance, suppose that Alice, a student at the Vrije Universiteit Amsterdam, writes the following SMS: “the professor assigned us several readings for homework”. We can use the five aforementioned concept to describe that the string “the professor” actually refers to Frank van Harmelen acting as a professor in the Vrije Universiteit Amsterdam, as follows (in Manchester Syntax)³:

```

Individual: ex:the-professor-string
Types:
  la:InformationEntity ,
  la:Expression
Facts:
  la:hasManifestation ex:the-professor-string-in-that-sms ,
  literal:hasLiteralValue "the professor" ,

```

³We reuse some entities from DBpedia (prefix *dbr*, base URL <http://dbpedia.org/resource/>), the *Literal Reification* ontology design pattern (prefix *literal*, base URL <http://www.essepuntato.it/2010/06/literalreification/>), the *Publishing Roles Ontology* (prefix *pro*, base URL <http://purl.org/spar/pro/>, and the *Scholarly Contributions and Roles Ontology* (prefix *scoro*, base URL <http://purl.org/spar/scoro/>).

```
la:denotes dbr:Frank_van_Harmelen ,
la:expresses ex:ProfessorAtVU
```

Individual: ex:the-professor-string-in-that-sms

Types:

```
la:InformationEntity ,
```

vla:Manifestation

Facts:

```
literal:hasLiteralValue "the professor" ,
```

```
la:denotes dbr:Frank_van_Harmelen ,
```

```
la:hasInterpretant ex:ProfessorAtVU
```

Individual: dbr:Frank_van_Harmelen

Types: la:Reference

Individual: ex:ProfessorAtVU

Types: la:Meaning

Class: ex:ProfessorAtVU

SubClassOf:

```
pro:holdsRoleInTime some (
```

```
pro:RoleInTime that
```

```
pro:withRole scoro:faculty-member and
```

```
pro:relatesToOrganization value
```

```
dbr:VU_University_Amsterdam)
```

Individual: ex:alice

Types: ar:Agent

Individual: ex:alice-writing-sms

Types:

```
sit:Situation ,
```

```
la:LinguisticAct
```

Facts:

```
sit:isSettingFor ex:the-professor-string ,
```

```
sit:isSettingFor ex:the-professor-string-in-that-sms ,
```

```
sit:isSettingFor dbr:Frank_van_Harmelen ,
```

```
sit:isSettingFor ex:ProfessorAtVU ,
```

```
sit:isSettingFor ex:alice
```

Among the five concepts introduced by this pattern, *Information entity* is the crucial one with respect to this chapter, and it is, thus, introduced in detail in Section 15.3.

15.2.2. Theoretical background

Different existing vocabularies tackle the representation of terms vs. meanings vs. things in general, e.g., SKOS [14], FRBR [16], CIDOC [13], WordNet-RDF [24], LMM [20], LMF [9], OntoLex-Lemon [11], etc. Each of these models has a particular approach depending on the original requirements they were designed

for (thesauri encoding, media item representation, standardizing digital library vocabularies, lexicon or (multi-)linguality representation, etc.). Thus, aligning all or part of them for a specific use is a difficult operation, specially when we consider the domain of document structures, where arbitrary representations lead to different realizations for the user, to lack of interoperability, and lock IE semantics into islands.

Semiotics is a theory that describes the core nature of most semantic phenomena. The ontology design pattern *Semiotics* that we describe here⁴ can be used to create interoperability among the above-mentioned vocabularies. In addition, it can be an off-the-shelf solution for each use case that needs to make semiotic distinctions.

Originally introduced by [4, 5], and further developed in other seminal works (we cite [6, 8, 22] as major examples of attempts at a systematic and practical design of the theory), semiotics introduces basic distinctions between entities that are arguments of a so-called *linguistic act* function (variously called in the literature as *sign function*, *speech act*, *communicative act*, *utterance*, *discourse*, *frame evocation*, etc.), which can be formalized as follows:

$$lact(e, m, r) \rightarrow s \quad (15.1)$$

where e is an *Information Entity* (variously called *expression*, *symbol*, *signifier*, etc.), m is a *Meaning* (variously called *concept*, *conceptualization*, *interpretant*, *sense*, *signified*, *conceptual frame*, etc.), r is a *Reference* (variously called *referent*, *object*, *entity*, *denotation*, etc.), and s is a *Situation* (variously called *state of affairs*, *state*, *conceptual frame occurrence*, etc.). For example, when someone says: *a dog is on the mat*, the word *dog* is an expression that *expresses* a meaning, e.g., a particular WordNet sense⁵, *denotes* a certain dog⁶, and, in the context of that linguistic act, contributes to communicate a state of the world⁷. The linguistic act function of *dog* within the above sentence may be therefore formalized as:

$$lact(dog, dog_{sense1}, Lola) \rightarrow LolaRoleInState1 \quad (15.2)$$

The *Semiotics* pattern introduced in the previous subsection is an OWL representation of the aforementioned linguistic act function.

15.3. Information Entities: objects and their realizations

15.3.1. The pattern

The *Information Realization* ontology pattern⁸ defines the concept *Information Object* and its relation with the concept *Information Realization*, as

⁴<http://ontologydesignpatterns.org/cp/owl/semiotics.owl>

⁵<http://www.ontologydesignpatterns.org/ont/wn/wn30/instances/synset-noun-dog-1>

⁶E.g., the dog “Lola”, identified by the resource <http://myontology.org/myanimals/Lola>

⁷Identified by another resource, e.g., <http://myontology.org/myanimals/LolaRoleInState1>

⁸<http://www.ontologydesignpatterns.org/cp/owl/informationrealization.owl>

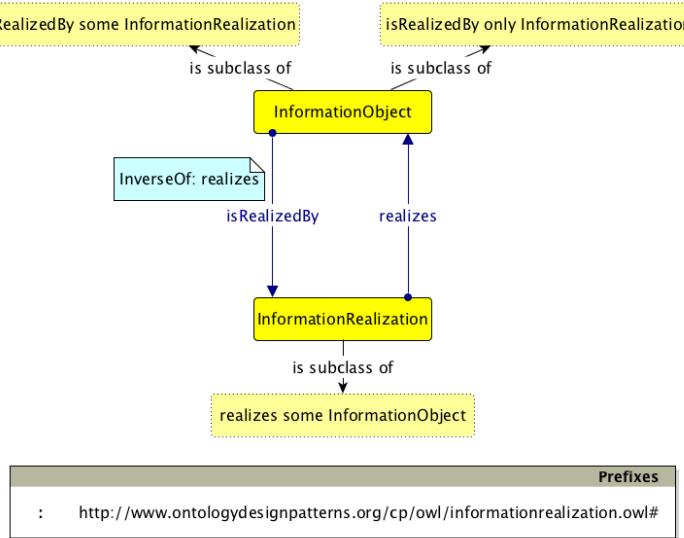


Figure 15.2. A Graffoo diagram representing the ontology pattern *information realization*.

shown in the Graffoo diagram in Fig. 15.2. In particular, these two concepts are extremely tied, since we cannot have a realization without its intended expression and vice versa, and are linked by means of the properties `realizes` (from `InformationRealization` to `InformationObject`) and its inverse (`isRealizedBy`).

Recalling the example presented in Section 15.2.1, where Alice wrote a text message to some friend after a University class, it is possible to distinguish between the actual SMS sent by Alice (i.e., the information realization) and the text of the message (i.e., the information object). Similarly, when a person sends an e-mail to several recipients, we have several realizations (each e-mail received by each recipient) of the same information object (i.e., the message contained in the e-mails). The latter example can be formalised in Manchester Syntax as follows:

```

Individual: ex:email-message
Types: InformationObject
Facts:
  isRealizedBy ex:email-message-sent-by-charline ,
  isRealizedBy ex:email-message-received-by-daniel ,
  isRealizedBy ex:email-message-received-by-eva
  
```

```

Individual: ex:email-message-sent-by-charline
Types: InformationRealization
Facts: realizes ex:email-message
  
```

```

Individual: ex:email-message-received-by-daniel
Types: InformationRealization
Facts: realizes ex:email-message
  
```

Individual: ex:email-message-received-by-eva
 Types: InformationRealization
 Facts: realizes ex:email-message

15.3.2. Theoretical background

Information entities (IE) include all (social) objects that are created and/or used by cognitive systems to communicate, reason, and stipulate new entities in a cognitive or social world. The patterns that are presented in this chapter are useful in any use case that includes IE as first-order entities, i.e., not just as labels or comments.

Any information entity is either an *Information Object*, or an *Information Realization*. The original distinction between information objects and their realizations lies in the ability to materialise, manifest, and replicate information across different media, based on which for example the copyright of the content of a published work is different from the copyright of the material book, of a specific copy of that book, of a lecture using that content, etc.

In more formal terms, according to DOLCE-Ultralight (DUL) (cf. [23] in this book), an information object is a particular kind of *social object*, i.e., a particular non-physical object carrying information, and created in the process of social communication – for instance, the (content of) this chapter, one of its previous drafts, the final publisher's *Version of Record*, etc. This concept maps to the class **1a:Expression**, i.e., the subclass of **1a:InformationEntity** introduced in Section 15.2.

On the other hand, an information realization is a particular physical object or event, i.e., an object that has a proper space region and that represents a particular materialisation of the information carried by an information object, or an event having such an object as participant. Generally speaking, a prototypical physical object of such kind has also an associated mass, but the nature of its mass can greatly vary based on the epistemological status of the object (scientifically measured, subjectively possible, imaginary, digital, etc.). For instance, a plausible realization of this paper seen as information object could be the actual printed copy of this chapter on a shelf, the PDF copy stored in a file system, and even the various copies of such PDF file one can make. This concept maps to the class **1a:Manifestation**, i.e., the subclass of **1a:InformationEntity** introduced in Section 15.2.

In the next sections we discuss how this pattern has been (explicitly or implicitly) used in the definition of new ontologies, or for the interoperability between existing ontologies proposed for the description of documents, bibliographic resources, and cultural heritage objects.

15.4. Manifestations of the Information Realization ontology pattern

As soon as an ontology introduces a way to distinguish between a piece of information, and the way it has been communicated to an audience, we can apply the

Information Realization pattern. In this section, we provide some examples from three different disciplines – namely Document Engineering, Library and Information Science, and Cultural Heritage – that show how this pattern is manifested in well-known international specifications and research works. A brief summary of such manifestation is shown in Fig. 15.3, where alignments between the patterns and the described models are provided.

15.4.1. Document Engineering: the case of markup semantics

Markup semantics is a very well-known and relevant issue for markup languages, and consequently for digital libraries. Nowadays, a large amount of content stored in digital libraries is encoded in XML, which provides a machine-readable mechanism for defining document structure by associating labels to fragments of text and/or other markup. While this association has a particular meaning (since each markup element asserts something about its content), what is asserted by the markup is not an issue of the markup itself. In fact, one of the goals of markup meta-languages is to avoid imposing any particular semantics: they express mere syntactic labels on the text, leaving the implicit semantics of the markup (some times defined with natural language definitions) to the interpretation of humans or tools programmed by a human.

In the past, we have experimented in providing some sort of formal mechanism to embed markup semantics within markup language schemas by means of Semantic Web technologies. The results of this experiment was the combined used of *EARMARK*⁹ with the linguistic acts ontology pattern introduced in 15.2 [1, 17]. EARMARK is an ontology of markup that makes explicit the implicit assumptions of markup languages (and, in particular, of the hierarchy of XML-based languages), providing a finer specification of the properties of markup, up to and including the possibility of toggling on and off the strict hierarchy of XML instantiations. Basically speaking, EARMARK can be used to define, in RDF, document markup items, such as elements and attributes, and their organisation, so as to provide a representation of any kind of markup actually stored in existing XML documents. Thus, having each markup item represented as individuals of the class **1a:InformationEntity** makes feasible the formal specification of a semantics it carries by means of the semiotics ontology pattern.

In addition to that, the specification of a markup item in EARMARK (i.e., an individual of the class **earmark:MarkupItem**, such as an element, and the way such element can be actually linearised in documents – such as in a Turtle file including the intended RDF statements for that element or a document containing the realization of that element in XML – relates well with information objects (i.e., **1a:Expression**) and information realizations (i.e., **1a:Manifestation**) respectively. Thus, from this perspective, the particular EARMARK element could be represented as pure information object that can be realised in different physical (i.e., digital) forms, each characterized by a specific format.

⁹<http://www.essepuntato.it/2008/12/earmark>

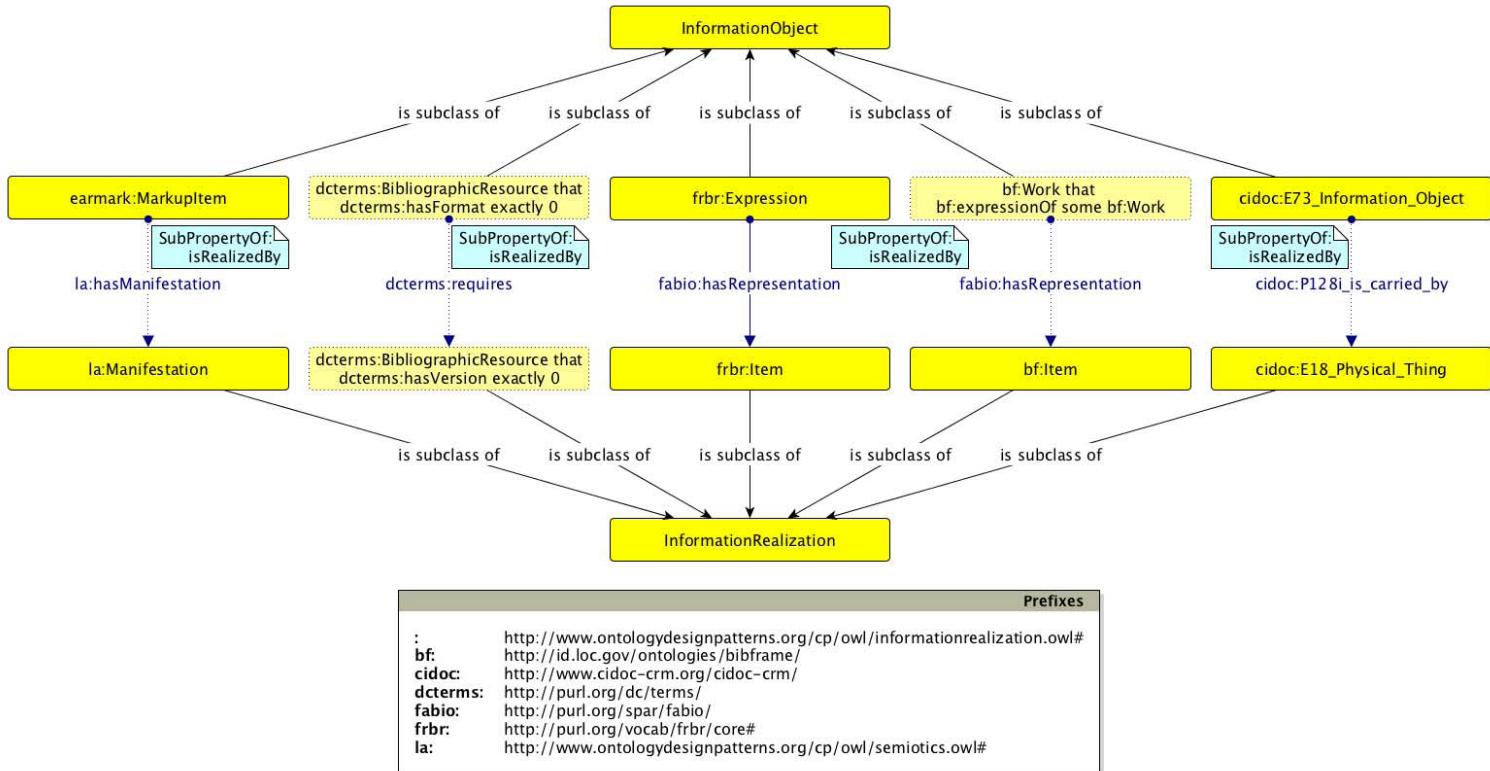


Figure 15.3. A Graffoo diagram that shows the alignments between existing models in the Document Engineering, Library and Information Science, and Cultural Heritage domains with the Information Realization ontology pattern.

15.4.2. Library and Information Science: metadata for bibliographic resources

In the past 30 years, several initiatives and works have been focused in the production of robust standards for providing descriptions of the metadata for bibliographic resources, such as articles, journals, books, and the like. Afterwards we discuss three of them – i.e., *Dublin Core Metadata Terms (DCTerms)*, the *Functional Requirements for Bibliographic Records (FRBR)*, and the *Bibliographic Framework (BIBFRAME)* – since they contain relevant manifestations of the *Information Realization* ontology pattern.

The current versions of *Dublin Core Metadata Terms (DCTerms)* [2] is the most widely used vocabulary for describing and cataloguing resources. This vocabulary has become particularly important and relevant for sharing metadata about documents among different repositories and digital libraries. It provides a general class, i.e., `dcterms:BibliographicResource`, for conveying any information related to bibliographic entities, which basically maps to `1a:InformationEntity`.

While there are no explicit classes defined for referring to information objects and related realizations, DCTerms makes available two pairs of properties that, implicitly, refer to these two aspects. First of all, we have the (object) properties `dcterms:hasVersion` and its inverse (`dcterms:isVersionOf`), which allows one to relate bibliographic resources when one is described as a (new) version of another in terms of their content, and regardless of the differences that may exist in format. These two properties refer to the information object level. On the other hand, the properties `dcterms:hasFormat` and its inverse (`dcterms:isFormatOf`) actually strictly relates two bibliographic resources that are substantially the same but in different format, which is a typical characteristic of information realizations.

Thus, while there are no explicit concepts for that, DCTerms separates, in some way, the layer defining information objects and the layer for information realizations by means of the aforementioned properties. Even if there is no explicit link between these two layers, one could use the property `dcterms:isRequiredBy` (defined as “the related resource that requires the described resource to support its function, delivery, or coherence”) or a more general one (e.g., `dcterms:relation`) for explicitly relating DCTerms-derived information objects with their realizations.

The *Functional Requirements for Bibliographic Record (FRBR)* [16] is a general model, proposed by the International Federation of Library Association (IFLA), for describing documents and their evolution. It works for both physical and digital resources and it has proved to be very flexible and powerful. FRBR describes all documents from four different and correlated points of view, i.e. the FRBR classes:

- Work, i.e., the high-level abstract Platonic concept of the essence of a distinct intellectual or artistic creation – e.g., what Lewis Carroll had in mind for *Alice's Adventure in Wonderland*;
- Expression¹⁰, i.e., identifying the actual content (e.g., the text) of the work

¹⁰Note that the meaning of this term is close to that of `1a:Expression` defined in the semiotics

- e.g., the actual content of *Alice's Adventure in Wonderlands* or its Italian translation provided by Luigi Lunari;
- Manifestation¹¹, i.e., the characteristics concerning the (e.g., digital or analog) format in which such expression is stored – e.g., the print volume of *Alice's Adventure in Wonderlands* published by Gribaudo Editore or the one in Kindle format made available by Amazon;
- Item¹², i.e., the particular physical or electronic copy of a work – e.g., the specific print copy of *Alice's Adventure in Wonderlands* one person has in her shelf, or the .mobi file in her e-book reader.

Concerning the information realization ontology pattern, only expressions and items provide actual informative content and can be the best candidates for mapping to information objects and information realization respectively, while the work and manifestation levels provide non-informative conceptualizations of a certain document and can be accessed by the other two layers. In fact, each work can have (one-to-many) realisations to expressions, each expression can have (one-to-many) embodiments into different manifestations, and each of the latter can have (one-to-many) exemplars.

While FRBR does not provide a direct “realization” link between an expression (information object) and the related item (information realization), some implementation of FRBR actually provides such facility. For instance, the *FRBR-aligned Bibliographic Ontology (FaBiO)* [18] defines the object property **fabio:hasRepresentation** that links directly the aforementioned FRBR classes¹³.

The *Bibliographic Frame (BIBFRAME, <http://id.loc.gov/ontologies/bibframe.rdf>)* is an OWL DL ontology that allows the description of bibliographic entities from a almost abstract level. It was initiated by the Library of Congress as a replacement for *MARC 21 Format for Bibliographic Data* [15], i.e., a very complex code for describing bibliographic resources as one of seven different primary types: book, continuing resource, computer file, maps, music, visual materials and mixed materials. In particular, one of the main goals of BIBFRAME concerns the study of possible transition paths for the MARC 21 formats, while preserving a robust data exchange that has supported resource sharing and cataloguing cost savings in recent decades.

The latest version of BIBFRAME (BF), released in April 2016, is based on three core classes, i.e., *BF Work* (describing the a conceptual essence of the cataloguing resource), *BF Instance* (one possible material embodiment of a BF Work), and *BF Item* (a physical or electronic) copy of a BF Instance. Basically speaking, the latter two classes map to the FRBR Manifestation and Item concepts, while the former one is an hybrid that includes, in some way, both the FRBR Work and the FRBR Expression – basically speaking, instances of the latter FRBR class are

ontology pattern.

¹¹Note that the meaning of this term is different from that of **la:Manifestation** defined in the linguistic acts ontology pattern.

¹²Note that the meaning of this term is close to that of **la:Manifestation** defined in the linguistic acts ontology pattern.

¹³It is worth noticing that **fabio:hasRepresentation** is actually defined as sub-property of the chain involving the two properties that link expressions to manifestations and manifestations to items respectively, i.e., **frbr:embodiment** o **frbr:exemplar**.

created in BIBFRAME when we have a BF Work that is actually an expression (property `bf:expressionOf`) of another BF Work. Thus, given this alignment, it is clear that the information realization ontology pattern is actually concretized by the concepts BF Work (being expression of another BF Work) and BF Item – leaving the class BF Instance as a way for specifying non-informative and mainly format-based characteristics of individuals from the class BF Item.

Also in this case, and similar to FRBR, there are no object properties that allow a direct link between the classes (expression of) BF Work and BF Item, leaving a concrete alignment with the property `isRealizedBy/realizes` unspecifiable directly within BIBFRAME. However, if we consider valid the alignment between the FRBR endeavours and the BF classes proposed above, `fabio:hasRepresentation` can be used for providing such a direct link.

15.4.3. Cultural Heritage: objects in museums

In the last years the Cultural Heritage domain has been one of the main areas of success for the development of metadata schemas and ontologies for the description of physical artefacts and related things. Among the various ontologies proposed, the most recognised one is undoubtedly the *ICOM's International Committee for Documentation (CIDOC) Conceptual Reference Model* [13]. According to the official website¹⁴, CIDOC-CRM “provides definitions and a formal structure for describing the implicit and explicit concepts and relationships used in cultural heritage documentation”, and it is already adopted by plenty of institutions as a standard format for sharing metadata, such as the British Museum¹⁵ and the Federico Zeri Foundation¹⁶ [3].

CIDOC-CRM explicitly applies the information realization ontology pattern to selected concepts of its taxonomy (available in RDFS at http://www.cidoc-crm.org/rdfs/cidoc_crm_v6.2.1-draft-b-2015October.rdfs). In particular, CIDOC-CRM has two classes, i.e., *E73 Information Object*, subclass of *E28 Conceptual Object*, and its realization *E84 Information Carrier*, subclass of *E19 Physical Object*. The individuals from the two classes can be linked by means of the property *P128i is carried by*.

15.5. Conclusions

This chapter has presented two foundational ontology design patterns: *Semiotics* and *Information Realization*, which are at the core (as direct reuse or originally implemented) of any design use case involving information entities. Each time an ontology designer needs to distinguish information from the world entities, or the concepts it is associated with, or if a use case requires to talk about information vs. its physical realization in different ways, the conceptualization behind the two patterns is evoked. They provide a shared lightweight axiomatisation, and

¹⁴<http://www.cidoc-crm.org>

¹⁵<http://collection.britishmuseum.org/>

¹⁶<http://data.fondazionezeri.unibo.it>

a vocabulary, to make ontologies involving information entities comparable and interoperable at the foundational level.

The current situation in the Semantic Web with respect to information entity modelling is quite heterogeneous. IEs are mostly included in textual fields as literals, which can only be searched in full text form if special indexing software is added to semantic technology platforms. On their turn, linguistic resources such as lexicons, gazetteers, and dictionaries are scarcely ported to linked data, so inhibiting their full exploitation jointly with factual data. Non-textual IEs such as music, images and videos have a limited representation counterpart in the Semantic Web: they are usually referred at the level of information objects, but actual content (realizations of music, pictures, videos, etc.) is mostly referenced as traditional Web 1.0 URIs, and not frequently annotated with semantic URIs denoting information objects.

This complex situation reveals a substantial limitation in the practical integration of semantic data and content on the open Semantic Web, which leaves to proprietary integration (iTunes and YouTube to mention the largest ones) the main operational implementation of multimedia semantics. The reason probably lies in the difficult management of copyrights and premium content from within open platforms, yet we believe that more could be done, as shown by experiences such as Jamendo, and IE patterns can boost the semantic integration of contents, tools, platforms, resources that are currently silo-ed or non-interoperable.

Bibliography

- [1] G. Barabucci, A. D. Iorio, S. Peroni, F. Poggi, and F. Vitali. Annotations with EARMARK in practice: a fairy tale. In *Proceedings of the 1st International Workshop on Collaborative Annotations in Shared Environment – metadata, vocabularies and techniques in the Digital Humanities, DH-CASE@DocEng 2013, Florence, Italy, September 10, 2013*, pages 11:1–11:8, 2013.
- [2] D. U. Board. DCMI Metadata Terms. Dcmi recommendation, Dublin Core Metadata Initiative, 2012.
- [3] M. Daquino, F. Mambelli, S. Peroni, F. Tomasi, and F. Vitali. Enhancing semantic expressivity in the cultural heritage domain: exposing the Zeri photo archive as linked open data. *CoRR*, abs/1605.01188, 2016.
- [4] F. de Saussure. *Cours de linguistique générale*. Payot, Lausanne, Suisse, 1906.
- [5] F. de Saussure. *Collected Papers of C.S. Peirce*. Harvard University Press, Cambridge, MA, 1958.
- [6] U. Eco. *Trattato di semiotica generale*. Bompiani, Milano, Italy, 1975.
- [7] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali. Modelling OWL ontologies with graffoo. In *The Semantic Web: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 320–325, 2014.
- [8] C. J. Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976.

- [9] G. Francopoulo, M. George, N. Calzolari, M. Monachini, N. Bel, M. Pet, C. Soria, et al. Lexical markup framework (LMF). In *Proceedings of LREC*, volume 6, 2006.
- [10] A. Gangemi and V. Presutti. Multi-layered n-ary patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. In this volume, chapter 6. IOS Press/AKA Verlag, 2016.
- [11] O. W. C. Group. OntoLex-Lemon: Final Model Specification, 2015.
- [12] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer. Integrating NLP using linked data. In *The Semantic Web-ISWC 2013*, pages 98–113. Springer, 2013.
- [13] ISO/TC 46/SC 4 Technical interoperability Working Group. ISO 21127:2014 - information and documentation – a reference ontology for the interchange of cultural heritage information. Iso standard, International Organization for Standardization, 2014.
- [14] A. Miles and S. Bechhofer. SKOS simple knowledge organization system reference, 2009.
- [15] L. of Congress Network Development and M. S. Office. MARK 21 format for bibliographic data. Specification document, Library of Congress, 2015.
- [16] I. S. G. on the FRBR. Functional requirements for bibliographic records - final report. Specification document, International Federation of Library Associations and Institutions, 2008.
- [17] S. Peroni, A. Gangemi, and F. Vitali. Dealing with markup semantics. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, pages 111–118, 2011.
- [18] S. Peroni and D. Shotton. Fabio and cito: Ontologies for describing bibliographic resources and citations. *J. Web Sem.*, 17:33–43, 2012.
- [19] S. Peroni and D. Shotton. Semantic Publishing and Referencing Ontologies, a.k.a. SPAR Ontologies, 2015.
- [20] D. Picca, A. M. Gliozzo, and A. Gangemi. LMM: an OWL-DL metamodel to represent heterogeneous lexical knowledge. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*, 2008.
- [21] V. Presutti, F. Draicchio, and A. Gangemi. Knowledge extraction based on discourse representation theory and linguistic frames. In *Knowledge Engineering and Knowledge Management*, pages 114–129. Springer, 2012.
- [22] J. R. Searle. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press, 1969.
- [23] Valentina and A. Gangemi. Dolce+D&S Ultralite and its main ontology design patterns. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. In this volume, chapter 5. IOS Press/AKA Verlag, 2016.
- [24] M. Van Assem, A. Gangemi, and G. Schreiber. Conversion of WordNet to a standard RDF/OWL representation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06), Genoa, Italy*, pages 237–242, 2006.

Chapter 16

The Role Patterns

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

16.1. Introduction

One rather common situation that one may encounter when setting up a schema for publishing (linked) data is modeling assertions such as “Nazifa is a student”, “Adila is an employee of Wright State University”, or “Barack is a president of United States”. Facing this scenario, a data publisher may simply go for a simple, albeit rather naive, solution by asserting that **Nazifa** is an instance of a class **Student**, **Adila** is an instance of a class **WSUEmployee**, and **Barack** is an instance of a class **USPresident**. When the data is published, it may serve its purpose at least in the beginning. However, the limit may soon be reached because of circumstances that cause those assertions no longer true, e.g., when Nazifa has graduated, Adila no longer works at Wright State University, or Barack finishes his final term as a president of United States. Of course, we can still say by then that **Nazifa was** a student, or **Adila was** an employee of Wright State University, or **Barack was** a president of United States.

If one were to inspect such situations rather carefully, one could see, for example, that the characteristic of “being a student” for Nazifa is something that is not always true in every situation. Rather, Nazifa is a student because she (currently) enrolls in some school. That is, there is an implicit relationship that underlies the characteristic of “being a student”. Similarly, Adila is an employee because he works for Wright State University, and well, Barack is a president of United States because he currently holds the office of United States president.

In ontology engineering, such a relationship is usually called *role*. The notion of role has been recognized as a rather fundamental notion that often appears in various modeling scenarios. Throughout this book, the notion of role pops up in several places. Presutti and Gangemi mentioned role [6] as one of the basic conceptualizations within DnS and DOLCE. Sales and Guizzardi focused their discussion regarding ontological anti-patterns on the notion of role [7]. Also,

Krisnadhi, et al. employed role to model the relationship between a chess game and the chess players, as well as the author(s) of the game report [4, 5]. There, they modeled, e.g., the white player of a chess game as a black player role that is performed by an agent, which is in this case, a person.

As aptly described by Sales and Guizzardi [7], the notion of role corresponds to a type that is both *anti-rigid* and *relationally dependent*. The former means that if an object holds/performs a role, there is at least one alternative situation in which it does not do so, while the latter means that an object performs a role only if it participates in a certain relation with another object and this relation may not necessarily hold in every possible situation. For example, being a student is a role that depends on the enrollment relation between the student and the school.

We do not intend, however, to present a deeper discussion regarding role and the foundational theory behind it. Rather, we aim to present a simple ontology design pattern that data publishers can readily use as part of the schema in publishing their data. The pattern is of course not entirely novel as there are patterns submitted to the ontologydesignpatterns.org portal, which capture some modeling variant of role. Presutti and Gangemi [6] mentioned Objectrole¹ and its specialization, AgentRole.² Additionally, the portal also contains the following patterns, which seem to capture the notion of role: ParticipantRole,³ Time indexed person role,⁴ and Roles (OWL 2)⁵ [3]. Nevertheless, having an explicit specification of the pattern spelled out in this book, which also reconciles those different proposals, could prove very useful for the reader.

16.2. Modeling and Formalization

We start with an observation that role is a type that captures a relationship. For example, saying that Nazifa is a student of WSU amounts to saying that Nazifa performs a student role in her relationship with WSU. From another perspective, we can also say that WSU *provides a student role* that is *performed by* Nazifa. The existence of such a student role implies the existence of “is a student of” relationship between Nazifa and WSU. We use the term “perform” or analogously, “play”, to indicate the object that holds an intuitively “active” part in the relationship, whereas the term “provides a role” is reserved for the object that holds the “passive” part in the relationship. In the example above, Nazifa, and not WSU, is the student in the relationship.

The next observation is that having an instance of role is meaningless if it exists by itself. A role must either be performed or provided by some object. The previous example illustrated a role that is provided by an object and performed by another object. In addition, example of a role that is provided by some object, but not performed by any object, is the driver role of an empty, non-autonomous

¹<http://ontologydesignpatterns.org/wiki/Submissions:Objectrole>

²<http://ontologydesignpatterns.org/wiki/Submissions:AgentRole>

³<http://ontologydesignpatterns.org/wiki/Submissions:ParticipantRole>

⁴http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_person_role

⁵http://ontologydesignpatterns.org/wiki/Submissions:Roles_%28OWL_2%29; and yes the name of the pattern is Roles (OWL 2) as given by the author who proposed it.

car, or the king role of a kingdom when its throne is vacant. Also, example of a role that is performed by an object, but not provided by any object, is the role that an old glass performed when it is used as a flower pot.

The third observation concerns the anti-rigidity nature of role. A precise definition of a role needs to express the condition in which an instance of the role exists, i.e., the underlying relationship becomes “true”. This may generally lead to severely complicated or extremely abstract definition of the role. To make the pattern less complex and easier to populate, we simply consider role to be time-indexed. That is, we use a temporal extent to specify the temporal boundary in which the underlying relationship of the role holds. Consequently, we can also understand the role pattern as a special case of multilayered n-ary patterns discussed by Gangemi and Presutti in [2].

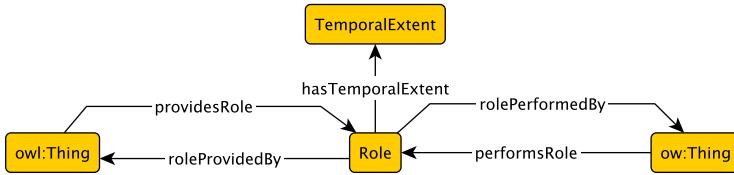
Our observations above are consistent with the following competency questions obtained from several existing patterns in ontologydesignpatterns.org portal mentioned in the previous section:

- what role does this object perform?
- which agent does play this role?
- what is the role that played by that agent?
- what role does this object play?
- which objects do play that role?
- what is the role of this object in this event?
- what is the object holding this role in this event?
- in what event did this object hold this role?

Note that we did not specifically mention “agent” and “event” in the discussion above. Nevertheless, agent can be understood as one among different kinds of object that can perform a role. We intentionally do not delve into the definition of agent, which can be too philosophical, and rather, leave it open for the users. In our chess examples (Chapter 1 [4] and 10 [5]), we considered a person to be an agent. Meanwhile, event is seen in this context as a kind of object that may provide a role. Of course, one could imagine that being able to provide a role would not be the only one characteristic that constitutes the definition of event. However, more detailed definition of event is beyond the scope of this chapter.

So, from our discussion above, we can see that in the Role pattern, we need a class that represents the notion of role, properties that connect the object that provides the role and the object that performs the role, and also, a placeholder that represents the temporal extent of the role and a property that connects it with the role. As a result, we obtain the generic Role pattern in Fig. 16.1.

The OWL 2 axioms for the generic Role pattern is given in 16.2, expressed in description logic notation [1]. Axiom (16.1), (16.2), (16.4), and (16.5) are either domain or range restrictions. Axiom (16.3) asserts inverse relationship between `providesAgentRole` and `roleProvidedBy`, while Axiom (16.6) asserts that between `rolePerformedBy` and `performsRole`. Note that the last two aforementioned axiom makes some of the domain and range restrictions above redundant, but we keep them anyway for the sake of readability. Axiom (16.7) contains a range restriction for `hasTemporalExtent` property as well as an existential assertion, which ensures that a role always have a temporal extent. Definition of `TemporalExtent` is left

**Figure 16.1.** Generic Role pattern

$$\top \sqsubseteq \forall \text{providesRole}. \text{Role} \quad (16.1)$$

$$\exists \text{roleProvidedBy}. \top \sqsubseteq \text{Role} \quad (16.2)$$

$$\text{providesRole} \equiv \text{roleProvidedBy}^{-} \quad (16.3)$$

$$\top \sqsubseteq \forall \text{performsRole}. \text{Role} \quad (16.4)$$

$$\exists \text{rolePerformedBy}. \top \sqsubseteq \text{Role} \quad (16.5)$$

$$\text{rolePerformedBy} \equiv \text{performsRole}^{-} \quad (16.6)$$

$$\begin{aligned} \text{Role} &\sqsubseteq \exists \text{hasTemporalExtent}. \text{TemporalExtent} \\ &\sqcap \forall \text{hasTemporalExtent}. \text{TemporalExtent} \\ &\sqcap (\leq 1 \text{ roleProvidedBy}. \top) \\ &\sqcap (\leq 1 \text{ rolePerformedBy}. \top) \end{aligned} \quad (16.7)$$

$$\text{Role} \sqsubseteq \exists \text{roleProvidedBy}. \top \sqcup \exists \text{rolePerformedBy}. \top \quad (16.8)$$

$$\text{DisjointClasses}(\text{Role}, \text{TemporalExtent}) \quad (16.9)$$

Figure 16.2. Axioms for generic Role pattern

open, but intuitively should include notions such as time instants and time intervals. This axiom also contains restrictions that no two objects can provide the same instance of `Role` and no two objects can perform the same instance of `Role`. In other words, one instance of `Role` cannot be shared by two different “role provider” nor can it be shared by two different “role performer”. Axiom (16.8) captures our observation that an instance of `Role` cannot exist by itself: either it is provided by an object or performed by one. Finally, Axiom (16.9) asserts the disjointness between `Role` and `TemporalExtent`.

The generic `Role` pattern can be specialized into Agent `Role` pattern as follows: we define `AgentRole` to be a subclass of `Role`, and then attach `Agent` as the range of `rolePerformedBy` and the domain of `performsRole`. The result is given in Fig. 16.3.

Axioms for Agent `Role` pattern is given in Fig. 16.4. Note that with the inverse relationship asserted in Axiom (16.6), both Axiom (16.11) and (16.12) entail the scoped domain and range restrictions for `performsRole` property in the context of Agent `Role` pattern.

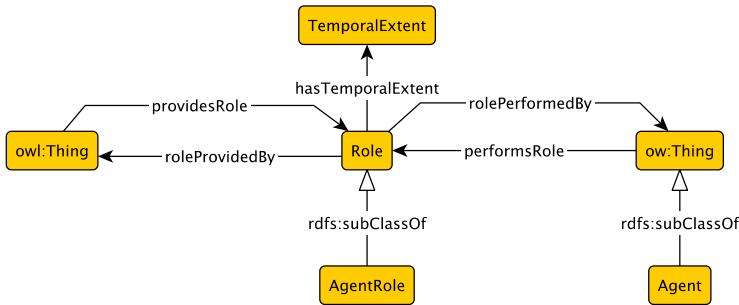


Figure 16.3. Agent Role pattern

import: (16.1), (16.2), (16.3), (16.4), (16.5), (16.6), (16.7), (16.8), (16.9)

$$\text{AgentRole} \sqsubseteq \text{Role} \quad (16.10)$$

$$\exists \text{rolePerformedBy}.\text{Agent} \sqsubseteq \text{AgentRole} \quad (16.11)$$

$$\text{AgentRole} \sqsubseteq \forall \text{rolePerformedBy}.\text{Agent} \quad (16.12)$$

Figure 16.4. Axioms for Agent Role pattern

16.3. Discussion

Given both the generic Role pattern and the Agent Role pattern from the preceding section, we can inspect and compare them with several patterns in ontologydesignpatterns.org portal that we mentioned in the beginning.

First, Objectrole pattern and its specialization, AgentRole pattern, correspond to half of the generic Role pattern and our Agent Role pattern, respectively. More precisely, Objectrole and AgentRole capture the relationship between objects and the roles they perform. In addition, although our patterns are time-indexed, they are still consistent with Objectrole and AgentRole since they do not formally exclude time information to be attached.

Next, ParticipantRole pattern can be seen as a specialization of the generic Role pattern where event acts as the provider of the role. The direct relationship between event and the participating object can be modeled based on our pattern by adding shortcut akin to what we did in Chapter 10 [5].

The Time indexed person role pattern appears to be an adorned specialization of the generic Role pattern where the role is performed by a person and the adornment comes in the form of the setting information for the role. Finally, the Roles (OWL 2) pattern – again, note that “Roles (OWL 2)” is indeed the name of the pattern given by the proposer – can also be seen as a variant of our Agent Role pattern.

Note that the aforementioned patterns from the ontologydesignpatterns.org portal have several competency questions as described at the beginning of previous section. After inspecting the generic Role pattern and the Agent Role pattern as

well as the above discussion, we argue that those competency questions can be satisfactorily answered by the two patterns defined in this chapter.

As a final remark, from the linked data publishing perspective, the generic Role pattern and Agent Role pattern specify a graph structure in which a relationship between two objects can be made temporalized. One instance of a role uniquely corresponds to the pair of objects for which the underlying relationship is asserted. For example, the “is student of” relationship between Nazifa and WSU can be captured by the Agent Role pattern whereby `StudentRole` is defined as a subclass of `AgentRole`, and furthermore, a unique instance of `StudentRole` is created for the pair of instances representing Nazifa and WSU respectively. This is exactly our approach in populating the Chess ontology as described in Chapter 10 [5]. In the meantime, we also see a potential for both patterns in this chapter to be used in helping data integration across different linked datasets because different properties occurring in the linked datasets represent factual assertions of relationships between objects and many of them can be covered as a specialization of the generic Role pattern or Agent Role pattern.

Acknowledgements. This work was partially supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*.

Bibliography

- [1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, New York, NY, USA, 2nd edition, 2010.
- [2] A. Gangemi and V. Presutti. Multi-layered n-ary patterns. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [3] R. Hoekstra. Representing social reality in OWL 2. In E. Sirin and K. Clark, editors, *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010), San Francisco, California, USA, June 21-22, 2010*, volume 614 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
- [4] A. Krisnadhi and P. Hitzler. Modeling with Ontology Design Patterns: Chess games as a worked example. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [5] A. Krisnadhi, N. Karima, P. Hitzler, R. Amini, M. Cheatham, V. Rodríguez-Doncel, and K. Janowicz. Ontology Design Patterns for linked data publishing. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.

- [6] V. Presutti and A. Gangemi. Dolce+DnS Ultralite and its main ontology design patterns. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.
- [7] T. P. Sales and G. Guizzardi. Ontological anti-patterns. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.

This page intentionally left blank

Chapter 17

The Semantic Trajectory Pattern

Yingjie Hu, STKO Lab, University of California Santa Barbara, USA

Krzysztof Janowicz, STKO Lab, University of California Santa Barbara, USA

17.1. Introduction and Motivation

Trajectory data have been widely used in a variety of domains, such as transportation, geography, ecology, oceanography, and zoology, to name but a few. These trajectories can be generated by humans, animals, cars, and other moving objects, and their location information can be recorded through different approaches including GPS receivers, RFID, WiFi, mobile phone cellular towers, and so forth. With the fast development of location-based technologies, it has become technically and economically feasible to collect a vast amount of trajectory data.

A trajectory can be considered as a set of temporally-indexed positions or *fixes*. Simplified, each *fix* can be represented by $\{x_i, y_i, t_i\}$, with x_i, y_i denoting a position in a 2D (geographic) space, and t_i representing a time point. While these spatiotemporal fixes can already support an exploration of the basic mobility pattern of moving objects, many applications require additional information to interpret the trajectories. For example, a transportation analysis based on the trajectories of vehicles may not be able to derive meaningful results, if the road network information or mode of transportation is not incorporated. Similarly, studies on the migration patterns of birds may require information about the studied birds, such as their species, body sizes, food sources, and age.

Semantic trajectories fill this gap by associating the fixes and segments (a linear interpolation between two consecutive fixes) of the trajectories with geographic information, domain knowledge, data provenance, as well as other related information. Such semantically enriched trajectories enhance the discovery of knowledge which otherwise may remain hidden. For example, the trajectories generated by humans can be better understood when the fixes are associated with corresponding place types, such as *restaurant* or *movie theater* and activities performed at these fixes, such as *dining* can be inferred based on the semantically annotated trajectories.

In this chapter, we present an ontology design pattern for semantic trajectories. While trajectory ontologies have been developed before [3, 4], they were confined to specific application areas and were not designed for the purpose of querying Linked Data, e.g., via the GeoSPARQL query language. This chapter is based on a previously published article [2]. In the following sections, we will introduce the competency questions that motivate this work (section 17.2) and then develop and formalize the semantic trajectory pattern using the Web Ontology Language (OWL) (section 17.3). We will then use an example trajectory dataset generated by an individual to demonstrate the applicability of this pattern (section 17.4). Finally, section 17.5 concludes this work by summarizing the characteristics of the semantic trajectory pattern.

17.2. Competency Questions

We motivate the development of the semantic trajectory pattern using competency questions. A competency question is a typical query that a domain expert may want to submit to a knowledge base to obtain answers [1]. To identify the competency questions for the semantic trajectory pattern, we held a two-day workshop to discuss with researchers from geography, ecology, computer science, and oceanography to understand the common queries they need. Such queries have been grouped into four categories, and are listed below. To better describe these queries, we will use specific examples, such as the trajectories generated by animals, while the semantic trajectory pattern is not restricted to these application areas.

Group 1: queries that can be answered by spatiotemporal fixes and segments.

- “Show birds which have stopped at x and y ”
- “Show birds which have moved at an average speed of 0.4 m/s”

Group 2: queries that need additional geographic information.

- “Show the trajectories that cross national parks”
- “Show the trajectories that have stopped by liquor stores”

Group 3: queries that need additional domain knowledge about the moving object.

- “Show the respective trajectories of sheepheads and kelp basses”
- “Show all trajectories that used multiple modes of transportation”

Group 4: queries that need additional information about the data provenance, e.g., the location-tracking device.

- “Show the trajectories captured by Gamin GPS”
- “Show the trajectories generated by iPhone users”

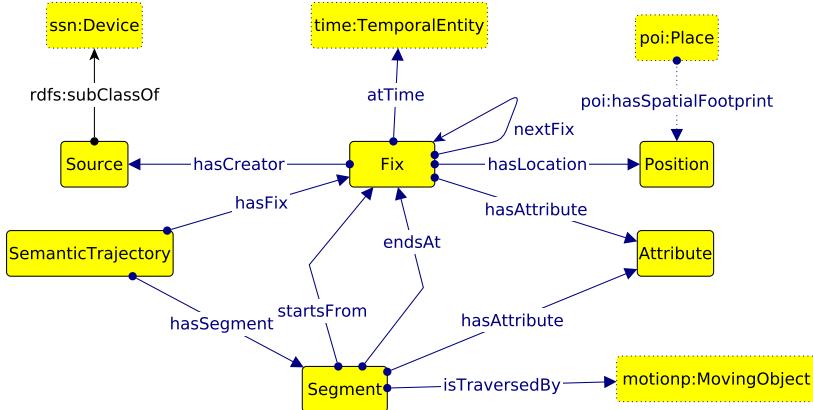


Figure 17.1. Schematical description of the pattern

17.3. Trajectory Pattern and its OWL Formalization

In this section, we present a semantic trajectory pattern that can answer the identified competency questions. Figure 17.1 shows a schematic view of the pattern. In the following paragraphs, we explain the classes and properties of the pattern respectively, and formally encode them using Web Ontology Language.

Fix. A *Fix* is a spatiotemporal point $\{x_i, y_i, t_i\}$ indicating the position of a moving object at an instant of time. Fixes are the atoms that constitute a trajectory. In Axiom 17.1, we require a fix to have a timestamp and a position and to belong to a trajectory.

$$\text{Fix} \sqsubseteq \exists \text{atTime}. \text{time:TemporalEntity} \sqcap \exists \text{hasLocation}. \text{Position} \\ \sqcap \exists \text{hasFix}^{-}. \text{SemanticTrajectory} \quad (17.1)$$

Segment. A *Segment* is a line connecting a starting fix $\{x_i, y_i, t_i\}$ and an ending fix $\{x_j, y_j, t_j\}$, with $t_i < t_j$. Since the moving subject can stay at the same position for a time period, $\{x_i, y_i\}$ may not necessarily be different from $\{x_j, y_j\}$. In Axiom 17.2, we enforce that every segment is connected to some fixes through the properties *startsWith* and *endsWith*. Axioms 17.3 and 17.4 enforce that every segment is connected to at most two fixes. Axiom 17.5 enforces that every segment belongs to a trajectory.

$$\text{Segment} \sqsubseteq \exists \text{startsWith}. \text{Fix} \sqcap \exists \text{endsWith}. \text{Fix} \quad (17.2)$$

$$\top \sqsubseteq \leq 1 \text{startsWith}. \top \quad (17.3)$$

$$\top \sqsubseteq \leq 1 \text{endsWith}. \top \quad (17.4)$$

$$\text{Segment} \sqsubseteq \exists \text{hasSegment}^{-}. \text{SemanticTrajectory} \quad (17.5)$$

time:TemporalEntity. This class expresses the temporal information associated with a fix. It serves as an interface which allows this ontology design

pattern to be aligned with other ontologies. This class can be implemented using *OWL-Time*.

Position. A *Position* is defined as a coordinate tuple $\{x_i, y_i\}$ which indicates the location of the fix. Similar to *time:TemporalEntity*, the class of *Position* acts as for interface to integrate the ontology design pattern with other ontologies on geographic information (e.g., the Point-of-Interest (POI) ontology).

Ordering Fixes within a Trajectory. We define the properties *hasNext*, *hasSuccessor*, *hasPrevious*, and *hasPredecessor* which can be automatically created using Axioms 17.6–17.10. These properties allow us to enforce an order among the fixes within a trajectory and such an order enables the verification of the temporal consistence of a set of fixes.

$$\text{startsFrom}^- \circ \text{endsAt} \sqsubseteq \text{hasNext} \quad (17.6)$$

$$\text{hasNext} \sqsubseteq \text{hasSuccessor} \quad (17.7)$$

$$\text{hasSuccessor} \circ \text{hasSuccessor} \sqsubseteq \text{hasSuccessor} \quad (17.8)$$

$$\text{hasNext}^- \sqsubseteq \text{hasPrevious} \quad (17.9)$$

$$\text{hasSuccessor}^- \sqsubseteq \text{hasPredecessor} \quad (17.10)$$

StartingFix, EndingFix, and Stop. *StartingFix*, *EndingFix*, and *Stop* are important classes which are necessary for executing some queries on trajectory data [3, 4]. While we do not explicitly define these classes in the ontology design pattern, Axioms 17.11 - 17.14 are proposed to automatically detect *StartingFix* and *EndingFix* as well as *StartingSegment* and *EndingSegment*. A *Stop* is a segment whose length is shorter than a distance threshold and whose duration is larger than a temporal threshold (both thresholds can be defined by the ontology user).

$$\text{Fix} \sqcap \neg \exists \text{endsAt}^-. \text{Segment} \sqsubseteq \text{StartingFix} \quad (17.11)$$

$$\text{Fix} \sqcap \neg \exists \text{startsFrom}^-. \text{Segment} \sqsubseteq \text{EndingFix} \quad (17.12)$$

$$\text{Segment} \sqcap \exists \text{startsFrom}. \text{StartingFix} \sqsubseteq \text{StartingSegment} \quad (17.13)$$

$$\text{Segment} \sqcap \exists \text{endsAt}. \text{EndingFix} \sqsubseteq \text{EndingSegment} \quad (17.14)$$

Attribute and hasAttribute. The class *Attribute* and the corresponding relation *hasAttribute* have been defined as the generic class and relation to connect fixes and segments to their attribute values, such as the speed at a particular fix or the bearing of a segment. Users of the pattern can either remain on this level or define their own subclasses and subroles, e.g., *hasSpeed.Speed*, based on the requirements of the particular applications.

Source. The *Source* class captures information about the device that has collected the trajectory data. Potential device information may include the manufacturer, produced year, spatial and temporal accuracies, product model, and so forth. Here, we recommend using the W3C SSN-XG ontology to provide sensor-related information.

isTraversedBy. This relation links a *Segment* with the corresponding moving subject. The *motionp:MovingObject* class is from the *Motion Pattern* developed in a previous GeoVoCamp and can be used as an interface for integrating domain knowledge with the trajectory data.

Semantic Trajectory. This class connects fixes, segments, and related knowledge into a meaningful path linking the origin and the destination. Using Axiom 17.15, we enforce that every trajectory is linked to at least one segment through the *hasSegment* property. Axioms 17.15 and 17.17 automatize the *hasFix* relationship linking every trajectory to the fixes within this trajectory.

$$\textit{SemanticTrajectory} \sqsubseteq \exists \textit{hasSegment}. \textit{Segment} \quad (17.15)$$

$$\textit{hasSegment} \circ \textit{startsWith} \sqsubseteq \textit{hasFix} \quad (17.16)$$

$$\textit{hasSegment} \circ \textit{endsAt} \sqsubseteq \textit{hasFix} \quad (17.17)$$

Domain and Ranges and Class Disjointness. We declare all classes defined for the pattern to be disjoint. We also recommend the definition of domains and ranges for existing classes. Axioms 17.18–17.21 show how to enforce some of these restrictions.

$$\exists \textit{hasSegment}. \textit{Segment} \sqsubseteq \textit{SemanticTrajectory} \quad (17.18)$$

$$\exists \textit{hasSegment}^{-}. \textit{SemanticTrajectory} \sqsubseteq \textit{Segment} \quad (17.19)$$

$$\exists \textit{hasFix}. \textit{Fix} \sqsubseteq \textit{SemanticTrajectory} \quad (17.20)$$

$$\exists \textit{hasFix}^{-}. \textit{SemanticTrajectory} \sqsubseteq \textit{Fix} \quad (17.21)$$

To sum up, the proposed semantic trajectory pattern uses *fixes* and *segments* to capture the trajectory data and defines a number of interfaces to incorporate additional geographic information, domain knowledge, and device data. The owl file of this pattern can be accessed at: <http://descartes-core.org/ontologies/trajectory/1.0/trajectory.owl>.

17.4. Application

In this section, we demonstrate the applicability of the proposed semantic trajectory pattern by using it to formally annotate an example trajectory dataset. This dataset was collected using a handheld GPS receiver, and it captured the trajectory of an individual who traveled to the 2012 GeoVoCamp held in Wright State University, Dayton, Ohio. During the trip, he also switched the transportation mode from driving to walking. Fragments of the formally annotated data are shown in Table 17.1. It can be seen from the table, a fix (e.g., *:fix1*) is associated with the location (*:pos1*), time (*:2012-09-15T11:26:22Z*), and device (*:mikesGPS*). A segment (e.g., *:segment1*) is associated with the fixes (*:fix1* and *:fix2*) and the moving subject (*:mikesCar*). Such integration allows us to answer competency questions, such as ‘show the trajectory segments that were traversed by Mike (walking) and Mike’s car (driving) respectively’ or ‘show the trajectory segments that were in Wright State University’. More details are given in [2].

Table 17.1. Part of the annotated data for the individual trajectory using N3

:mikesTrajectory	a	:SemanticTrajectory;
	:hasSegment	:segment1, :segment2, ...;
	:hasFix	:fix1, :fix2, :fix3, :fix4, ...;
:mikesCar	a	motionp:MovingObject;
:mikesGPS	a	:Source;
:segment1	a	:Segment;
	:startsFrom	:fix1;
	:endsAt	:fix2;
	:isTraversedBy	:mikesCar;
:fix1	a	:Fix;
	:hasCreator	:mikesGPS;
	:inXSDDataTime	:2012-09-15T11:26:22Z;
	:hasLocation	:pos1;
:pos1	a	:Position;
	geo:asWKT	Point(x_0, y_0);

17.5. Conclusions

In this paper we presented an ontology design pattern for semantic trajectories and highlighted its applicability. The pattern can be used to semantically annotate trajectory data from a range of domains. This pattern has the following characteristics: 1) **Expressiveness**: the design pattern can express a trajectory's spatiotemporal properties, geographic knowledge, domain knowledge, as well as relations among them. 2) **Simplicity**: only a minimal number of classes and relations are defined, which makes the design pattern easy to understand, reuse, and extend. 3) **Flexibility**: the provided interfaces (such as *Source*) allow the ODP user to integrate additional knowledge according to the specific needs of the application. 4) **Scalability**: depending on the required granularity of the applications, the ontology design pattern can model trajectories at different scales. While we have demonstrated this pattern using the trajectory data from an individual, this ODP can also be applied to annotating the trajectory data from other moving subjects, such as vehicles and animals.

Bibliography

- [1] M. Grüniger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5*, volume 7, pages 212–221, 1994.
- [2] Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitze, M. Dean, and D. Kolas. A geo-ontology design pattern for semantic trajectories. In *Spatial Information Theory*, pages 438–456. Springer, 2013.
- [3] S. Spaccapietra, C. Parent, M. Damiani, J. De Macedo, F. Porto, and

- C. Vangenot. A conceptual view on trajectories. *Data and Knowledge Engineering*, 65(1):126–146, 2008.
- [4] Z. Yan, J. Macedo, C. Parent, and S. Spaccapietra. Trajectory ontologies and queries. *Transactions in GIS*, 12:75–91, 2008.

This page intentionally left blank

Chapter 18

An Ontology Design Pattern for Detector Final States

David Carral, Wright State University

18.1. Introduction

Particle Physics, the study of the fundamental building blocks and forces of our universe, involves some of the largest experimental apparatus ever constructed, like the ALICE, ATLAS, CMS, and LHCb experiments located at the Large Hadron Collider (LHC) at CERN. Each of these “experiments” is a very large collaboration of physicists who work as a team to design, build, and operate the particle detectors and to produce measurements characterizing the particles that make up the universe. The measurements are inherently statistical in nature: often billions or trillions of particle collisions are analyzed to determine probabilities or probability densities associated with a given physical process. Because many of the experiments collect multi-purpose data, careful attention must be paid to defining the measurement that is to be made.

Despite the many thousands of papers published since the advent of particle physics in the 1940s, the field has no formal way of representing or classifying experimental results – no *metadata* accompanies an article to formally describe the physics result therein. A number of scenarios would be enabled with such a representation. For example, a physicist from ATLAS, or a theorist, could search an external database for previous work done by CMS in order to compare results. Even a physicist inside ATLAS could search an internal database for previous examples similar to a planned analysis; a substantial amount of time and effort can be saved by starting from some preexisting work.

We intend to address this situation with our ontology design pattern. Results in particle physics take many forms, but all are based on the selection of a target set of characteristics, a *detector final state* that defines the ingredients of the measurement. The fundamental unit of S is the individual interaction of a set of particles, or an “event.” An event could, for example, be captured from a single interaction of counter-rotating particles in a collider or from the collision

of a high-energy cosmic ray in the atmosphere. The selection characteristics refer to properties of an event and can describe the presence or absence of specific particles observed by the detector in the aftermath of the collision, or potentially more global properties of the products produced in the collision, such as the total energy released. Since the physics results we wish to describe and preserve in a repository are all based on the selection of *one or more* detector final states, this is a necessary ingredient of an ontology covering the whole particle physics analysis life cycle.

As an example, consider the case where the products of a collision coalesce into two narrow cone-like “jets” of particles, typical of the production of two high energy quarks. This would be a “two-jet final” state. A short list of measurements involving just two particle jets in the detector might include: a cross section, or rate, for finding two particle jets in a collision; a search for a new particle that decays to two particle jets; and the angular separation as a function of energy of two particle jets. Many measurements can be made beginning with a given final state, but the selection of the final state is the basis for each analysis. Many other examples of detector final states are possible, of course. Having a formal set of descriptors of this selection is thus essential to enabling easy access to archived experimental results.

Competency questions have been recognized as a good approach to detect and generalize the modeling requirements from multiple domains that an ontology can represent. They are queries that a domain expert would be expected to run against a knowledge base. For the proposed final state ODP, such competency questions include:

1. Retrieve all analyses requiring particles to have an invariant mass near the Z pole.
2. Retrieve all analyses that used jets in the final state.
3. Retrieve all analyses that veto extra leptons .
4. Retrieve all analyses requiring large missing energy.

18.2. Formalization

This section presents the detector final state pattern by discussing the more interesting classes, properties, and axioms. Description Logics (DL) [2] notation is used to present the axioms. To encode the pattern, we make use of the logic fragment *SROIQ* as defined in [5], which is the basis for the OWL 2 DL standard [4]. The proposed ODP has been formally encoded using the Web Ontology Language (OWL).¹ A schematic view of the pattern is shown in Figure 18.1.

DetectorFinalState:

The main purpose of this pattern is to characterize the information that is contained in some existing publication (Article). Thus, every detector final state

¹The pattern can be downloaded from <http://daselab.org/publications/final-state-detector-odp-owl-ontology>.

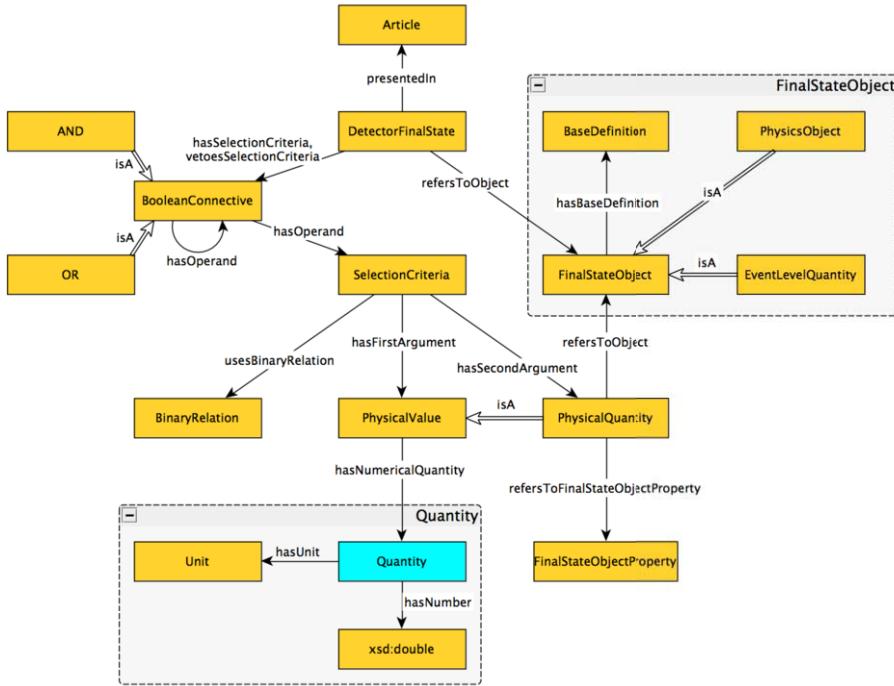


Figure 18.1. A schematic view of the DetectorFinalState ODP

(DetectorFinalState) must be associated with some article, and every article is associated with some detector final state. Such a requirement is explicitly enforced by the following axioms.

$$\text{DetectorFinalState} \sqsubseteq \exists \text{presentedIn}.\text{Article} \quad (18.1)$$

$$\text{Article} \sqsubseteq \exists \text{presentedIn}.\text{DetectorFinalState} \quad (18.2)$$

A detector final state (DetectorFinalState) formally describes and structures information about a physics analysis (measurement) that is defined by its use of a common set of particle physics characteristics. As such, it must describe those characteristics of the fundamental “event” that have been selected to make the measurement. It is defined, amongst other features, by a set of particles/objects (PhysicsObjects) contained in the event and by global quantities formed by performing some operation on the ensemble of objects contained in such an event (EventLevelQuantity). Both particles/objects and the ensemble measurements are referred to in this pattern as final state objects (FinalStateObject).

With the following axioms, we enforce that (18.3) every detector final state must refer to at least one final state object, (18.4) all final state objects are either event level quantities or physics objects and (18.4) all event level quantities and

physics objects are final state objects.

$$\text{DetectorFinalState} \sqsubseteq \exists \text{referstTo.FinalStateObject} \quad (18.3)$$

$$\text{FinalStateObject} \equiv \text{EventLevelQuantity} \sqcup \text{PhysicsObject} \quad (18.4)$$

In order to select events of interest, these objects are subject to selection criteria that are used to define a collection of events that serves as the basis for a physics measurement, and hence must be captured in the pattern. A detector final state (`DetectorFinalState`) then conveys numerical information describing the selection. This numerical information is referred as the selection criteria (`SelectionCriteria`) which models a complex boolean set of unary and binary restrictions. We make use of the classes `And` and `Or` to define complex selection criteria.

$$\text{DetectorFinalState} \sqsubseteq \exists \text{hasSelectionCriteria.}(\text{SelectionCriteria} \sqcup \text{And} \sqcup \text{Or}) \quad (18.5)$$

$$\text{And} \sqsubseteq \exists \text{hasOperand.}(\text{SelectionCriteria} \sqcup \text{And} \sqcup \text{Or}) \quad (18.6)$$

$$\text{Or} \sqsubseteq \exists \text{hasOperand.}(\text{SelectionCriteria} \sqcup \text{And} \sqcup \text{Or}) \quad (18.7)$$

FinalStateObject:

As mentioned above, there are two different types of final state objects in our model: physics objects and event level quantities. Each of these is defined by a restricted vocabulary and will point to another class, namely `BaseDefinition`, which will serve as a hook to provide more specific information about these types. Axiomatically, then, every `PhysicsObject` and every `EventLevelQuantity` are `FinalStateObjects`:

$$\text{PhysicsObject} \sqsubseteq \text{FinalStateObject} \quad (18.8)$$

$$\text{EventLevelQuantity} \sqsubseteq \text{FinalStateObject} \quad (18.9)$$

In order for these quantities to have meaning, each of the `FinalStateObjects` requires a `BaseDefinition` that describes the criteria for the creation of the `FinalStateObject`.

An example typical selection could be “retrieve all detector final states involving some electron with $p_T > 40$ GeV.” In this case, the selection requires a particular type of final state object and has a restriction of 40 GeV, a `PhysicalValue` with a `NumericalValue` of 40 and a `Unit` of GeV, on the `PhysicalQuantity` p_T , which is shorthand for the momentum of a particle in the plane transverse to the beam axis. In general, a `SelectionCriteria` must indeed have a `FirstArgument` specifying a value and a `SecondArgument` specifying of which `PhysicalQuantity` this value is an instance, with some sort of binary operator (< or >, for example) specifying the desired relationship.

A small sample of RDF data which instantiates the pattern can be found at <http://daselab.org/publications/final-state-detector-odp-rdf-data>. We not only include terminological axioms in our ontology but also populate the pattern using data from existing publications [1].

Acknowledgements This chapter is a revised version of [3], a paper presented at the 2015 Workshop on Ontology and Semantic Web Patterns. The author acknowledges funding by the National Science Foundation under award 1440202 *EarthCube building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*.

Bibliography

- [1] G. Aad et al. Search for pair-produced long-lived neutral particles decaying in the ATLAS hadronic calorimeter in pp collisions at $\sqrt{s} = 8$ TeV. *Phys.Lett.*, B743:15–34, 2015.
- [2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.
- [3] D. Carral, M. Cheatham, S. Dallmeier-Tiessen, P. Herterich, M. D. Hildreth, P. Hitzler, A. Krisnadhi, K. Lassila-Perini, E. Sexton-Kennedy, C. Vardeman, and G. Watts. An ontology design pattern for particle physics analysis. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.
- [4] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at <http://www.w3.org/TR/owl2-primer/>.
- [5] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible \mathcal{SRQI} . In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

This page intentionally left blank

Appendix

This page intentionally left blank

Appendix A

A Primer on RDF and OWL

Frederick Maier, University of Georgia, Athens, GA

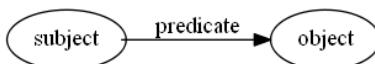
This appendix provides a brief introduction to RDF [12, 18] and OWL [19, 31], the primary modelling formalisms used in Semantic Web applications. The essential structure of RDF graphs is presented, followed by discussions of the concrete RDF syntax Turtle, RDFS, and the semantics of RDF. OWL, its semantics, and tractable OWL profiles are then presented.

Though they were both specifically developed for the Semantic Web and their development occurred in tandem, the motivations underlying RDF and OWL are different, and this is reflected in the syntax and semantics of each. RDF is intended to be simple and open, allowing users on the Web to make statements about resources. From the standpoint of logical expressivity, it is relatively limited. OWL, in contrast, is intended explicitly for the development of formal ontologies, and as such it has a more sophisticated logical syntax and semantics.

Both RDF and OWL are developed and maintained by the World Wide Web Consortium (W3C).¹ Since this appendix provides only an incomplete overview of the formalisms, readers are encouraged to review the W3C specifications.

A.1. RDF: The Resource Description Framework

The **Resource Description Framework (RDF)** is a graph-based formalism for making statements about *resources*. RDF makes no restriction on what a resource can be—a resource is simply something worth talking about. An *RDF graph* is a set of zero or more *RDF statements*, where each statement is a *subject predicate object* triple. Each triple forms a simple directed graph. An RDF graph is a superimposition of these.



¹<https://www.w3.org/>

Subjects and objects form nodes in the graph, and predicates (called *properties*) form directed edges. Logically, each RDF graph is a conjunction of statements.

The most recent version of RDF, RDF 1.1, [12], was published in 2014. An earlier version (sometimes called RDF 1.0), appeared in 2004 [18]. While the syntax and semantics of the two are mostly the same, there are differences, and so readers should be aware of both.

The discussion below focuses on RDF 1.1. In particular, it focuses on its canonical *abstract syntax* [12] which defines the basic structure of RDF 1.1 graphs. The abstract syntax is distinct from the several *concrete syntaxes* [9–11, 14, 26, 28] that exist and which can be used to encode the same abstract graph. When written, an encoding of an RDF graph is called an *RDF document*.

A.1.1. RDF Abstract Syntax

A.1.1.1. RDF Statements

An RDF statement is composed of *RDF terms*, where each term is an *internationalized resource identifier (IRI)* [13], a *blank node*, or a *literal*. The types of term are disjoint, and there are restrictions on where each can appear in a statement.

1. A subject must be an IRI or blank node.
2. A predicate must be an IRI.
3. An object must be an IRI, blank node, or literal.

IRIs are character strings that refer to resources by name (they label nodes and edges in an RDF graph). Blank nodes, in contrast, indicate resources that are unknown or have no name. They are analogous to existentially quantified variables in first order logic. A literal (used to refer to numbers and character strings, for instance) is a character string together with a *datatype* IRI (in place of a datatype IRI, a *language tag* might appear).

A.1.1.2. Internationalized Resource Identifiers (IRIs)

IRIs are generalizations of *Uniform Resource Identifiers* (URIs) [5], the difference being that URIs are restricted to using US-ASCII characters, while IRIs allow characters from the larger Unicode character set. The syntaxes for both URIs and IRIs are defined in documents published by the Internet Engineering Task Force [5, 13]. Both “URI” and “IRI” are encountered in the research literature, but when discussing RDF, the terms can mostly be used interchangeably.

Though not essential to understanding RDF, IRIs have the hierarchical structure shown below.²

scheme ":" ["/" authority] path ["?" query] ["#" fragment]

²Readers should refer to the IETF documents [5, 13] for a complete description of the structure of IRIs and URIs.

Table A.1. Common namespace IRIs and prefixes

<i>prefix</i>	<i>namespace IRI</i>	<i>description</i>
<code>rdf</code>	<code>http://www.w3.org/1999/02/22-rdf-syntax-ns#</code>	<i>RDF Concepts</i>
<code>rdfs</code>	<code>http://www.w3.org/2000/01/rdf-schema#</code>	<i>RDF Schema</i>
<code>xsd</code>	<code>http://www.w3.org/2001/XMLSchema#</code>	<i>XML Schema</i>
<code>owl</code>	<code>http://www.w3.org/2002/07/owl#</code>	<i>OWL Schema</i>

The italicized elements refer to parts of the IRI, and the elements in square brackets are optional. An IRI begins with a *scheme* (`http` and `ftp` are common examples), followed by a colon. Each scheme can place further restrictions on identifiers defined under it. The optional *authority* is intended to permit delegation of processing IRIs to some entity. The *path* component consists of a sequence of segments separated by occurrences of ‘/’.³

Both the *query* and *fragment* are optional. In Web address IRIs, the query typically consists of attribute-value pairs (e.g., for encoding form data). The fragment sometimes specifies a secondary resource (relative to the one identified by the other elements of the IRI). Nevertheless, two IRIs having the same scheme and authority but distinct fragments or paths constitute distinct IRIs.

Although the hierarchical structure of IRIs can sometimes be used to locate or access the resources they denote (in which case we say that the IRI can be *dereferenced*), there is no requirement that IRIs support this functionality. IRIs are best viewed as proper names.

A.1.1.3. Relative vs Absolute IRIs

The hierarchical nature of IRIs allows grouping resources under a common prefix. RDF 1.1’s abstract syntax, however, requires all IRIs to be *absolute* (fully specified). Nevertheless, some concrete syntaxes permit specifying a *base* IRI. Afterwards, *relative IRIs* omitting the common prefix can be defined. E.g., in Turtle [26], once `<http://example.org/>` has been specified as the base IRI, the relative IRI `<Animal>` can be used in place of `<http://example.org/Animal>`. It must be possible to *resolve* a relative IRI against the base IRI, making it absolute.

It is also possible in certain syntaxes to designate IRIs as *namespace* IRIs and associate each with a short *prefix* (this is based on XML namespaces [6, 7]). A *prefixed name* can then be used to identify a resource. For instance, in Turtle, if `ex` is associated with `http://example.org`, then the prefixed name `ex:Animal` can be used. Some common namespace IRIs and prefixes are given in Table A.1.

A.1.1.4. Literals

A literal in RDF 1.1. consists of a *lexical form* (a Unicode string) together with a *datatype* IRI. Furthermore, if and only if the datatype is `rdf:langString`, a nonempty IETF language tag [25] must be present (in which case, the literal is called a *language-tagged string*). It is common to write literals using the form

`"value"^^datatypeIRI [@languageTag].`

³If the authority is present, then the path is either empty or else begins with “/”. If there is no authority, then the path begins with “/” but mustn’t begin with “//”.

Since language tags are only used with `rdf:langString`, the datatype IRI is often omitted in language-tagged strings.

RDF uses several built-in XML Schema datatypes (e.g., `xsd:boolean`, `xsd:string`, `xsd:integer`, `xsd:double`) [24]. In addition, RDF defines `rdf:HTML` and `rdf:XMLLiteral` for representing HTML and XML data as literals.

Example 1. Example literals.

```
"hello world"^^xsd:string "2002-09-24"^^xsd:date
"hello"@en                      "-4"^^xsd:integer
```

Formally, a datatype consists of a *lexical space* (a set of Unicode strings), a *value space* (the space of possible values), and a mapping from the first to the second. The mapping can be viewed as providing an interpretation of the strings in the lexical space. A datatype may be designated by a *datatype IRI*. Systems which process RDF are not required to recognize any datatype IRIs—those the system does recognize are simply called *recognized datatype IRIs*.

A.1.1.5. Blank Nodes

Graphically, a blank node is a node without a label. Some concrete syntaxes nevertheless permit associating a blank node with a *blank node identifier*. This allows statements about the same entity to be separated in an RDF document. The identifiers are specific to the concrete syntaxes, however, and are not part of the RDF abstract syntax. Also, since the scope of an identifier is local to the document in which it appears, care must be taken when merging RDF documents containing blank node identifiers. If two documents to be merged use the same blank node identifier, then the identifier in one document should be renamed.

A.1.1.6. RDF Datasets

An *RDF dataset* consists of a single unnamed *default graph* together with 0 or more graphs named by IRIs. In the context of a dataset, the scope of a blank node identifier is the whole dataset (meaning that the dataset's graphs share identifiers). RDF datasets were introduced in RDF 1.1.

A.1.1.7. Graph and Dataset Isomorphism

Comparison of RDF graphs and datasets is typically done at the syntactic level. This is complicated by the presence of blank nodes (which, again, may be viewed as existentially quantified variables). The relevant notions of equivalence are *graph isomorphism* and *dataset isomorphism* [12]. RDF graphs G and G' are isomorphic if there is a bijective function from terms of G to terms of G' such that each IRI and literal of G is mapped to itself, each blank node of G is mapped to a blank node of G' , and a triple is in G if and only if the triple obtained after mapping its terms is in G' . Dataset isomorphism is defined similarly.

Table A.2. W3C Concrete RDF Syntaxes

<i>concrete syntax</i>	<i>description</i>
RDF 1.1 XML Syntax [14]	An XML-based syntax for RDF.
RDF 1.1 Turtle [26]	A concise, readable syntax for RDF.
RDF 1.1 N-Triples [9]	Graphs are written as ‘.’-delimited sequence of triples.
RDF 1.1 N-Quads [11]	Extends N-Triples with a 4th argument.
RDF 1.1 TriG [10]	Extends Turtle to permit encoding of RDF datasets.
RDF 1.1 JSON-LD [28]	Encodes RDF statements using JSON syntax.

A.1.2. Turtle: Terse RDF Triple Language

Several so-called *concrete syntaxes* exist for writing, storing, and transporting RDF graphs and datasets. When a graph or dataset is written using one of these, it is called an *RDF document*. Some concrete syntaxes specified by the W3C are listed in Table A.2. The original specification of RDF used XML as its canonical serialization syntax. With RDF 1.1, however, the XML syntax is one among many. Turtle (*Terse RDF Triple Language*) [26] is a popular and fairly human-readable syntax, and so we will present it now and use it in later sections to describe additional features of RDF.

A Turtle document is a UTF-8 encoded character string consisting of 0 or more statements. Such a statement is either a *directive* (for specifying base and prefix IRIs) or else a textual representation of RDF statements. Both types of Turtle statement use RDF terms, which in Turtle have the following format.

- *IRIs*: An absolute or relative IRI enclosed in <...>, or else a prefixed name *prefix:local*.
- *Literals*: "value" or "value"^^*IRI* or "value"@*lang*, where *lang* is a language tag and *IRI* is a datatype IRI in one of the forms noted above.
- *Blank nodes*: _:label (e.g., _:x) or []. In a given document, multiple occurrences of _:x represent the same blank node, while multiple occurrences of [] represent distinct blank nodes.

A.1.2.1. Directives

Base and prefix IRIs are specified using @base and @prefix, respectively, or BASE and PREFIX (the latter are used for compatibility with SPARQL [32]). An ending period (‘.’) is required for @base and @prefix directives but omitted in directives beginning with BASE and PREFIX.

Example 2. A set of directives beginning a Turtle document.

```
@base <http://example.org/> . # the base IRI
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Above, # by itself begins a comment (which terminates at the end of the line).

A.1.2.2. RDF Statements

RDF statements in Turtle may be written as simple triples, each ending with a period ‘.’. However, a collection of triples can be written more concisely using *object lists* and *predicate lists*.

- **simple triples:** a *subject predicate object*.
- **object lists:** *subject predicate objectList*, where *objectList* is a comma-delimited list of object RDF terms.
- **predicate lists:** *subject predicateList*, where *predicateList* is a semicolon-delimited list of *predicate objectList* pairs.

Above, *subject* is an IRI or blank node, *predicate* is an IRI, and *object* is an IRI, blank node, or literal.

Example 3. Three different Turtle encodings of the same triple.

```
@base  <http://example.org/>.
@prefix ex: <http://example.org/>.

<http://example.org/mary>           # Encoding 1
  <http://example.org/hasChild>
    <http://example.org/louis>.

ex:mary ex:hasChild ex:louis.      # Encoding 2
<mary> <hasChild> <louis>.        # Encoding 3
```

Example 4. Triples involving predicate and object lists. Here, John has three children, one of them anonymous. Jeanne has one child and two parents.

```
@prefix      ex: <http://example.org/> .
ex:john     ex:hasChild     ex:charles, ex:philip, _:x .
ex:jeanne   ex:hasChild     ex:catherine ;
            ex:hasParent   ex:marguerite, ex:henry .
```

A.1.2.3. Nesting Statements in Blank Nodes

In a given Turtle document, multiple occurrences of [] represent distinct blank nodes. Nevertheless, property lists can be nested inside of [...], with the blank node serving as the subject of the statements appearing within.

Example 5. Here, John has a child, and someone loves both Rob and Jane.

```
ex:john ex:hasChild [] .
[ex:loves ex:rob ; ex:loves ex:jane ].
```

A.1.2.4. Literals

Literals in Turtle are typically quoted and may be followed by a language tag or datatype IRI (but not both). If a language tag is used, then it must be preceded with “@”. A datatype tag must be preceded with “^^”. If no datatype is specified, then the datatype is interpreted as xsd:string.

Example 6. Statements involving literals.

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:hobbes ex:wrote _:abook .
_:abook ex:title "Leviathan"@en ;
ex:publishDate "1651"^^xsd:gYear .
```

A.1.2.5. Collections

RDF provides syntax to specify lists of resources, similar to the way lists are represented in the programming languages Prolog and Lisp. A list consists of a *head* or first element (indicated by property `rdf:first`) and a *tail* (`rdf:rest`). The tail can either be another list or else an empty list token (`rdf:nil`).

In Turtle, lists are written as space-delimited term sequences enclosed in parentheses (...). The sequence can be empty, with () mapping to `rdf:nil`.

Example 7. A statement relating `john` to a collection of names.

```
ex:john ex:aliases ("johnathan" "john" "jon" "j") .
```

A.1.3. RDF Schema

RDF provides a small vocabulary⁴ for making statements about resources. *RDF Schema* (RDFS) [8] extends this vocabulary, allowing one to define not only classes, properties, and their instances, but also subclass and subproperty hierarchies as well as restrictions on the domain and range of properties.

A.1.3.1. Classes

Classes and class instances are specified using `rdf:type` and `rdfs:Class`. To say that one class is a subclass of another, `rdfs:subClassOf` is used. This allows the creation of a class hierarchy. There is a topmost class, `rdfs:Resource`. All resources are instances of it, and all classes are subclasses of it. Table A.3 lists classes predefined in RDFS.

Example 8.

```
ex:Human rdf:type rdfs:Class .
ex:bruce rdf:type ex:Human .
ex:Human rdfs:subClassOf ex:Animal .
```

`rdf:type` can be read as “is an instance of”. In Example 8, `ex:Human` is an instance of `rdfs:Class`, and `ex:bruce` is an instance of class `ex:Human`. Note that a class can be an instance of a class, including itself.

⁴The vocabulary is defined in <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

Table A.3. Predefined RDFS classes [8].

<i>class name</i>	<i>description</i>
rdfs:Resource	all resources
rdfs:Literal	all literal values
rdf:langString	language-tagged strings
rdf:HTML	HTML literal values
rdf:XMLLiteral	XML literal values
rdfs:Class	all classes
rdf:Property	all RDF properties
rdfs:Datatype	all RDF datatypes
rdf:Statement	all RDF statements
rdf:Bag	unordered containers
rdf:Seq	ordered containers
rdf:Alt	containers of alternatives
rdfs:Container	all RDF containers
rdfs:ContainerMembershipProperty	container properties: rdf:_1, rdf:_2, etc.
rdf:List	RDF lists

A.1.3.2. Properties

To say that a resource is a property, one uses `rdf:type` and `rdf:Property`. To specify subproperty relationships, one uses `rdfs:subPropertyOf`. Table A.4 lists properties predefined in RDFS.

Example 9. Specifying properties and subproperties.

```
ex:hasMother  rdf:type          rdf:Property .
ex:hasParent  rdf:type          rdf:Property .
ex:hasMother  rdfs:subPropertyOf ex:hasParent .
```

A.1.3.3. Containers and Collections

Collections (`rdf>List`) were mentioned in the discussion of Turtle. Their head-tail structure imposes an order on lists, and there is a mechanism to delineate the end of the list (`rdf:nil`). Other kinds of groups can also be defined. An `rdfs:Container` is open in the sense that additional elements can always be asserted to belong to it. `rdf:Bag`, `rdf:Seq`, and `rdf:Alt` are all subclasses of it. Though indistinguishable from the standpoint of the formal semantics for RDF, the first two are used to stand for unordered and ordered containers, respectively, while the last is used to indicate alternatives. In each, one asserts membership using the properties `rdf:_1`, `rdf:_2`, ... (e.g., `ex:a rdf:_4 ex:C`). Each of these is a subproperty of `rdfs:member`.

A.1.3.4. Reification

RDF provides vocabulary for taking *about* RDF statements (i.e., for *reifying* them).

Example 10. Reifying a statement.

Table A.4. Predefined RDFS properties [8]. Each property `rdf:_n` is a subproperty of `rdfs:member` and an instance of `rdfs:ContainerMembershipProperty`.

property name	description	domain	range
X <code>rdf:type</code> Y	X is an instance of class Y.	<code>rdfs:Resource</code>	<code>rdfs:Class</code>
X <code>rdfs:subClassOf</code> Y	X is an subclass of Y.	<code>rdfs:Class</code>	<code>rdfs:Class</code>
X <code>rdfs:subPropertyOf</code> Y	X is an subproperty of Y.	<code>rdf:Property</code>	<code>rdf:Property</code>
X <code>rdfs:domain</code> Y	The domain of X is Y.	<code>rdf:Property</code>	<code>rdfs:Class</code>
X <code>rdfs:range</code> Y	The range of X is Y.	<code>rdf:Property</code>	<code>rdfs:Class</code>
X <code>rdfs:label</code> Y	Specifies a label Y for X.	<code>rdfs:Resource</code>	<code>rdfs:Literal</code>
X <code>rdfs:comment</code> Y	Specifies a comment Y for X.	<code>rdfs:Resource</code>	<code>rdfs:Literal</code>
X <code>rdfs:member</code> Y	Y is a member of X.	<code>rdfs:Resource</code>	<code>rdfs:Resource</code>
X <code>rdf:first</code> Y	Y is the first element of list X.	<code>rdf:List</code>	<code>rdfs:Resource</code>
X <code>rdf:rest</code> Y	Y is the tail of list X.	<code>rdf:List</code>	<code>rdf:List</code>
X <code>rdfs:seeAlso</code> Y	Y provides information on X.	<code>rdfs:Resource</code>	<code>rdfs:Resource</code>
X <code>rdfs:isDefinedBy</code> Y	Y is the definition of X.	<code>rdfs:Resource</code>	<code>rdfs:Resource</code>
X <code>rdf:value</code> Y	Y is the value of X.	<code>rdfs:Resource</code>	<code>rdfs:Resource</code>
X <code>rdf:subject</code> Y	Y is statement X's subject.	<code>rdf:Statement</code>	<code>rdfs:Resource</code>
X <code>rdf:predicate</code> Y	Y is statement X's predicate.	<code>rdf:Statement</code>	<code>rdfs:Resource</code>
X <code>rdf:object</code> Y	Y is statement X's object.	<code>rdf:Statement</code>	<code>rdfs:Resource</code>
X <code>rdf:_n</code> Y	Y is n th member of X ($n \geq 1$).	<code>rdf:Statement</code>	<code>rdfs:Resource</code>

```

ex:triple1 rdf:type      rdf:Statement.
ex:triple1 rdf:subject    ex:superman .
ex:triple1 rdf:predicate  rdf:type .
ex:triple1 rdf:object     ex:FlyingThing.

```

Once a resource has been identified as a statement, additional statements can be made about it (e.g., `ex:lois ex:believes ex:triple1`). Note that this is not the same as the statement appearing as a triple in the graph. To talk about a statement is different than asserting its truth.

A.1.3.5. Domain and Range Restrictions

A property is a binary relationship, and one may assert triples in RDFS to specify the property's domain and range. When multiple assertions are made, the domain or range is inferred to be the intersection of the classes specified.

Example 11. Restricting the domain and range of `ex:hasMother`.

```

ex:hasMother rdfs:range   ex:Person .
ex:hasMother rdfs:range   ex:Female .
ex:hasMother rdfs:domain  ex:Person .
ex:jeanne     ex:hasMother  ex:marguerite .

```

In Example 11, it should be possible to infer that `ex:marguerite` is an instance of both `ex:Person` and `ex:Female`, and also that `ex:jeanne` is an instance of the class `ex:Person`. Furthermore, given the assertions in Example 9, it should be possible to infer `ex:jeanne ex:hasParent ex:marguerite`. The semantics of RDFS in fact allows all of these conclusions to be drawn.

A.1.4. RDF Semantics

RDF has a model theoretic semantics similar to that of classical first order logic. An interpretation first specifies a domain of discourse and then interprets the terms of each RDF statement relative to that chosen domain. This ultimately provides a truth value for each RDF statement. An RDF graph G *entails* another graph G' if every interpretation making G true also makes G' true.

RDF in fact specifies multiple related semantics, each providing its own account of entailment and differentiated in large part by the manner in which datatypes and IRIs in the RDF and RDFS namespaces are handled. We discuss the simplest semantics below. Note, however, that in the various semantics (as in first order logic), a *unique name assumption* is not made. In an interpretation, distinct IRIs can refer to same element of the domain.

As defined in [15], a *simple interpretation* I consists of the following:

1. A nonempty set IR of resources—the *domain of discourse*.
2. A set IP of properties of I (which can overlap with IR).
3. A function $IEXT$ from IP to $IR \times IR$.
4. A function IS from IRIs to $IR \cup IP$.
5. A (possibly partial) function IL from literals to IR .

For graphs without blank nodes (i.e., *ground* graphs), the values of expressions are given below:

1. If E is a literal, then $I(E) = IL(E)$.
2. If E is a IRI, then $I(E) = IS(E)$.
3. If E is a triple $s p o$, then $I(E) = \text{true}$ if and only if $I(p) \in IP$ and $\langle I(s), I(o) \rangle \in IEXT(I(p))$. Otherwise, $I(E) = \text{false}$.
4. If E is a ground graph, then $I(E) = \text{false}$ if there exists a triple E' in E such that $I(E') = \text{false}$. Otherwise $I(E) = \text{true}$.

For graphs with blank nodes, interpretations are augmented with a mapping A from blank nodes to IR , forming a combined interpretation $[I + A]$ for all RDF terms. Under this scheme, $I(E) = \text{true}$ if $[I + A](E) = \text{true}$ for some mapping A , and $I(E) = \text{false}$ otherwise.

An interpretation I *satisfies* E (I is a model of E) if and only if $I(E) = \text{true}$. A graph G *simply entails* a graph E if and only if every model of G is one of E .

Example 12. For graphs

$$G_1 = \{\text{ex:a ex:p ex:b.}\}, G_2 = \{_x \text{ ex:p } _y.\}, \text{ and } G_3 = \{_z \text{ ex:p } _z.\},$$

no matter what resources `ex:a` and `ex:b` are mapped to, it is clear that we can construct an assignment mapping `_x` and `_y` to the same resources. According to the account of simple entailment above, G_1 simply entails G_2 . However, G_1 does not simply entail G_3 . If `ex:a` and `ex:b` are mapped to different resources, then no assignment can simultaneously map `_z` to both.

Table A.5. Axiomatic triples for RDF and RDFS entailment [15].*RDF axiomatic triples*

These assert that `rdf:type`, `rdf:subject`, `rdf:predicate`, `rdf:object`, `rdf:first`, `rdf:rest`, `rdf:value`, and `rdf:_1` are all instances of `rdf:Property`, and that `rdf:nil` is a instance of `rdf>List`.

RDFS axiomatic triples

<code>rdf:type</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Class.</code>
<code>rdfs:domain</code>	<code>rdfs:domain rdf:Property ; rdfs:range rdfs:Class.</code>
<code>rdfs:range</code>	<code>rdfs:domain rdf:Property ; rdfs:range rdfs:Class.</code>
<code>rdfs:subPropertyOf</code>	<code>rdfs:domain rdf:Property ; rdfs:range rdf:Property.</code>
<code>rdfs:subClassOf</code>	<code>rdfs:domain rdfs:Class ; rdfs:range rdfs:Class.</code>
<code>rdf:subject</code>	<code>rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource.</code>
<code>rdf:predicate</code>	<code>rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource.</code>
<code>rdf:object</code>	<code>rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource.</code>
<code>rdfs:member</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource.</code>
<code>rdf:first</code>	<code>rdfs:domain rdf>List ; rdfs:range rdfs:Resource.</code>
<code>rdf:rest</code>	<code>rdfs:domain rdf>List ; rdfs:range rdf>List.</code>
<code>rdfs:seeAlso</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource.</code>
<code>rdfs:isDefinedBy</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource.</code>
<code>rdfs:comment</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Literal.</code>
<code>rdfs:label</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Literal.</code>
<code>rdf:value</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource.</code>
<code>rdf:Alt</code>	<code>rdfs:subClassOf rdfs:Container .</code>
<code>rdf:Bag</code>	<code>rdfs:subClassOf rdfs:Container .</code>
<code>rdf:Seq</code>	<code>rdfs:subClassOf rdfs:Container .</code>
<code>rdfs:isDefinedBy</code>	<code>rdfs:subPropertyOf rdfs:seeAlso .</code>
<code>rdfs:Datatype</code>	<code>rdfs:subClassOf rdfs:Class .</code>
<code>rdfs:ContainerMembershipProperty</code>	<code>rdfs:subClassOf rdf:Property .</code>
<code>rdf:_n</code>	<code>rdf:type rdfs:ContainerMembershipProperty .</code>
<code>rdf:_n</code>	<code>rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .</code>

The other accounts of entailment—*datatype entailment*, *RDF entailment*, and *RDFS entailment*—place further restrictions on interpretations. Ultimately, these extensions ensure that statements involving literals and those such as

```
ex:Human rdfs:subClassOf ex:Animal .
```

permit the inferences they are intended to.

We will not discuss the other semantics in detail. However, we note that in valid RDF and RDFS interpretations, there are sets of *axiomatic triples* that must be true. These are shown in Table A.5. Furthermore, we note that entailment in RDF and RDFS can be captured using the *entailment rules* in Table A.6. These allow the use of simple inference procedures (e.g., forward chaining) to draw conclusions from RDF graphs. The inference procedures are easily implemented and can form the basis of an automated reasoner. As written, the rules permit sound inferences. They are not complete, but they can be made so by allowing literals in the subject position of statements and blank nodes in the predicate position [15].⁵

⁵The rules in Table A.6 are taken from [15]. Original results are due to ter Horst [29, 30].

Table A.6. Sound RDF, RDFS entailment rules [15]. Here, D is a set of datatypes.

	<i>If</i>	<i>Then</i>
rdfD1	$x \ p \ s^{\sim}d . \ (\text{where } d \text{ in } D)$	$x \ p \ _:n . \ _:n \ \text{rdf:type } d .$
rdfD2	$x \ p \ y .$	$p \ \text{rdf:type } \text{rdf:Property} .$
rdfs1	any IRI a in D	$a \ \text{rdf:type } \text{rdfs:Datatype} .$
rdfs2	$a \ \text{rdfs:domain } x .$ $y \ a \ z .$	$y \ \text{rdf:type } x .$
rdfs3	$a \ \text{rdfs:range } x .$ $y \ a \ z .$	$z \ \text{rdf:type } x .$
rdfs4a	$x \ a \ y .$	$x \ \text{rdf:type } \text{rdfs:Resource} .$
rdfs4b	$x \ a \ y .$	$y \ \text{rdf:type } \text{rdfs:Resource} .$
rdfs5	$x \ \text{rdfs:subPropertyOf } y .$ $y \ \text{rdfs:subPropertyOf } z .$	$x \ \text{rdfs:subPropertyOf } z .$
rdfs6	$x \ \text{rdf:type } \text{rdf:Property} .$	$x \ \text{rdfs:subPropertyOf } x .$
rdfs7	$a \ \text{rdfs:subPropertyOf } b .$ $x \ a \ y .$	$x \ b \ y .$
rdfs8	$x \ \text{rdf:type } \text{rdfs:Class} .$	$x \ \text{rdfs:subClassOf } \text{rdfs:Resource} .$
rdfs9	$x \ \text{rdfs:subClassOf } y .$ $z \ \text{rdf:type } x .$	$z \ \text{rdf:type } y .$
rdfs10	$x \ \text{rdf:type } \text{rdfs:Class} .$	$x \ \text{rdfs:subClassOf } x .$
rdfs11	$x \ \text{rdfs:subClassOf } y .$ $y \ \text{rdfs:subClassOf } z .$	$x \ \text{rdfs:subClassOf } z .$
rdfs12	$x \ \text{rdf:type }$ $\text{rdfs:ContainerMembershipProperty} .$	$x \ \text{rdfs:subPropertyOf } \text{rdfs:member} .$
rdfs13	$x \ \text{rdf:type } \text{rdfs:Datatype} .$	$x \ \text{rdfs:subClassOf } \text{rdfs:Literal} .$

A.2. OWL: The Web Ontology Language

With RDFS, one can define simple class and property hierarchies, and the formal accounts of entailment allow drawing logical conclusions from what is explicitly represented in a graph. As a knowledge representation framework, however, RDFS is limited. For instance, while it is possible to say in RDFS that `Triangle` and `Square` are classes, it is not possible to say that they have no instances in common. RDFS has no effective way of expressing logical negation. In contrast, OWL provides significantly more expressiveness. As in RDFS, assertions can be made about resources, and class and property hierarchies can be defined. Additionally, symbols exist in OWL for expressing negation, conjunction, disjunction, and forms of existential and universal quantification. These and other features allow more sophisticated assertions than is possible in RDFS.

An ontology in OWL can be viewed as a set of logical statements about some domain of interest. It is common to divide the statements into two general classes: *terminological axioms* specify the general nature of the domain (defining classes, properties, and their interrelationships), while instance-level *assertions* make claims about particular individuals. The terminological axioms of an ontology are said to constitute its *TBox*, while the instance-level assertions constitute its *ABox*. Also, some authors use the term “ontology” to refer to a TBox alone and reserve “knowledge base” to refer to a combination of TBox and ABox. We will not make that distinction here, however.

OWL 2 [31] is the most recent version of OWL. It extends a previous version sometimes called OWL 1, published in 2004 [19]. Several different concrete

syntaxes exist for OWL 2. The RDF syntaxes mentioned earlier can be used, but there is also a separate *OWL/XML* syntax [21] that closely resembles OWL’s formal specification. There are also two syntaxes—the Manchester Syntax [16] and the Functional Syntax [23], which are more easily understood by humans.

A.2.1. OWL 2 Structure and Functional Syntax

The formal structure of OWL 2 is defined by a set of UML diagrams [23]. However, the concrete functional syntax is closely based on the UML specification, and so we will present OWL using it.

In the functional syntax, an OWL 2 ontology document has the form shown in Table A.7. A document consists of a sequence of prefix declarations (relating prefix names to IRIs), followed by an ontology encoding. The ontology encoding itself consists of optional IRIs naming the ontology and providing a version for it, a set of import statements (for importing axioms from other ontologies), a set of annotations (providing further information about the ontology), and a set of axioms forming the logical core of the ontology. In the following discussion, we will not discuss annotations or imports, instead focussing on the logical core.

A.2.1.1. Entities

The fundamental elements of OWL are *named individuals*, *classes*, and *properties*. These are all *entities* in OWL, as are datatypes. All are named by IRIs. To declare that an IRI refers to an entity of a given type, a declaration axiom (as shown in Table A.7) is used. OWL distinguishes between *object properties*, which relate individuals to individuals, and *data properties*, which relate individuals to literals. A declaration axiom can declare an IRI to be of either type.

In addition to entities, *literals* are also defined, as are *anonymous individuals*. In the functional syntax, the identifiers for anonymous individuals have the form $_ : X$. As in serialized RDF, the scope of an identifier is local to the OWL ontology in which it appears.

A.2.1.2. Class Expressions

A class can be named using an IRI, and OWL provides names for two predefined classes:

- `owl:Thing`, which stands for the class of all individuals; and
- `owl:Nothing`, which stands for the empty class.

Named classes and various *class constructors* are used to create complex class expressions (see Table A.8). Intuitively, OWL class expressions describe sets of individuals. `owl:Thing` represents the set of all things, and `owl:Nothing` represents the empty set.

A.2.1.3. Basic Set Operations

The simplest class constructors correspond to basic operations on sets: `ObjectComplementOf`, `ObjectUnionOf`, and `ObjectIntersectionOf`. Using `ObjectOneOf`, one can also define a class by explicitly enumerating its elements.

Example 13.

```
ObjectUnionOf(ex:Bird ex:Cat ex:Dog)
ObjectIntersectionOf(ObjectComplementOf(ex:Red) ex:Car)
ObjectOneOf(ex:jeanne ex:marguerite ex:henry)
```

The first class expression above describes the class of all birds, cats, and dogs. The second describes the class of non-red cars. The third describes the class consisting of (at most) three things: Jeanne, Marguerite, and Henry.

Example 14.

```
ObjectUnionOf(ex:Square ObjectComplementOf(ex:Square))
ObjectIntersectionOf(ex:Square ObjectComplementOf(ex:Square))
```

Here, the first expression describes the class of all things that are either square or not square (that is, everything), while the second describes things that are simultaneously square and not square (that is, nothing). These are equivalent to `owl:Thing` and `owl:Nothing`, respectively.

A.2.1.4. Existential and Universal Restrictions

Existential and universal restrictions on both object and data properties can also be specified.

Example 15.

```
ObjectSomeValuesFrom(ex:hasChild ex:Physicist)
ObjectAllValuesFrom(ex:hasChild ex:Physicist)
```

The first of these describes the class of things related to at least one physicist via the `hasChild` property. The second describes the class of things related to *only* physicists via `hasChild` (anyone with a non-physicist child is excluded).

A.2.1.5. Cardinality Restrictions

Cardinality restrictions on properties restrict the number of things an individual is related to via the property.

Example 16.

```
ObjectMinCardinality(1 ex:hasChild ex:Physicist)
ObjectMaxCardinality(3 ex:hasChild)
ObjectExactCardinality(2 ex:owns ex:Dog)
```

The first class expression above describes the class of things having at least one child who is a physicist. The second describes the class of things with no more than 3 children (it is equivalent to `ObjectMaxCardinality(3 ex:hasChild owl:Thing)`). The third specifies the class of things that own exactly 2 dogs. Note that the last argument in a cardinality restriction is optional. If present, then the restriction is called a *qualified cardinality restriction*. If absent, the restriction is an *unqualified cardinality restriction*.

A.2.1.6. Individual Value and Self Restrictions

An *individual value restriction* is used to describe a class of things related to a particular individual via a property expression. Similarly, a *self restriction* is used to describe a class of individuals related to themselves via the property expression.

Example 17. The first of these describes the class of things employed by John. The second describes the class of things that are self-employed.

```
ObjectHasValue(ex:employedBy ex:john)
ObjectHasSelf(ex:employedBy)
```

A.2.1.7. Property Expressions

As noted earlier, OWL distinguishes between object properties and data properties. For each type, OWL defines both universal and empty properties.

- `owl:topObjectProperty`: relates all individuals to all individuals.
- `owl:topDataProperty`: relates all individuals to all literals.
- `owl:bottomObjectProperty`: relates no individuals.
- `owl:bottomDataProperty`: relates no individual to any literal.

The first two can be thought of as consisting of all possible pairs of the appropriate type. The latter two can be thought of as empty.

An OWL *object property expression* is either an object property IRI or else an expression of the form `ObjectInverseOf(IRI)`. The latter defines the pairs that are the inverses of members of the object property named by *IRI*. Inverses are not defined for data properties, and so a data property IRI is the only kind of *data property expression*.

A.2.1.8. Literals, Data Ranges, and Datatype Definitions

Typed literals in the OWL functional syntax have the form `value^^datatype`, where *value* is a quoted string and *datatype* is a datatype identifier. Plain literals are also allowed, with or without language tags (e.g., "hello"@en).

One may also define *data ranges*, which are sets of tuples of literals. A datatype is a data range having arity 1. Counterparts to basic class operations are defined (`DataComplementOf`, `DataUnionOf`, `DataIntersectionOf`, `DataOneOf`), and these can be used in property restrictions. A list of data range expressions is shown in Table A.8.

Example 18. A data range corresponding to `{"3"^^xsd:integer}`.

```
DataIntersectionOf(
  DataOneOf("1"^^xsd:integer "2"^^xsd:integer "3"^^xsd:integer)
  DataOneOf("3"^^xsd:integer "4"^^xsd:integer "5"^^xsd:integer))
```

A *datatype definition* associates a data range with an IRI. As shown in Example 19, an IRI-literal pair such as `xsd:minInclusive "90"8xsd:integer` is called a *constraining facet*, and it is used in a datatype restriction to limit the permitted values of an underlying datatype (in this case, `xsd:integer`).

Example 19. The datatype `ex:normalValue` is defined to be `xsd:integer` values in the interval 90 (inclusive) to 110 (exclusive).

```
DatatypeDefinition( ex:normalValue
  DatatypeRestriction( xsd:integer
    xsd:minInclusive "90"^^xsd:integer
    xsd:maxExclusive "110"^^xsd:integer))
```

A.2.1.9. Class Axioms

As shown in Table A.7, there are several types of axiom in OWL. A class hierarchy can be defined in OWL using `SubClassOf`. In addition, one can assert that two or more classes are equivalent (`EquivalentClasses`) or disjoint (`DisjointClasses`).

Example 20. Simple class axioms.

```
SubClassOf(ex:Cat ex:Animal)
EquivalentClasses(ex:Cat ex:Feline)
DisjointClasses(ex:Cat ex:Dog ex:Bird)
```

It's also possible in OWL to assert (using `DisjointUnion`) that a class is equivalent to the union of pairwise disjoint classes.

Example 21. The axiom `DisjointUnion(ex:Integer ex:Even ex:Odd)` is equivalent to the two axioms shown below.

```
EquivalentClasses(ex:Integer ObjectUnionOf(ex:Even ex:Odd))
DisjointClasses(ex:Even ex:Odd)
```

A.2.1.10. Property Axioms

Parallelling the axioms for class expressions, OWL allows one to assert that one property is a subproperty of another, and also that two or more properties are equivalent or disjoint. One can also specify the domain and range for a property.

This applies to both object properties and data properties. There are differences, however. In a `SubObjectPropertyOf` axiom, the first argument can be a *property chain*. A chain $p_1 p_2 \dots p_n$ can be interpreted in first order logic as $p_1(x_0, x_1) \wedge p_2(x_1, x_2) \wedge \dots \wedge p_n(x_{n-1}, x_n)$.

Example 22. A subproperty axiom involving a property chain.

```
SubObjectPropertyOf(
  ObjectPropertyChain(ex:brotherOf ex:motherOf)
  ex:uncleOf)
```

This is equivalent to the first order implication shown below.

$$\forall x \forall y \forall z [brotherOf(x, y) \wedge motherOf(y, z) \Rightarrow uncleOf(x, z)]$$

Furthermore, in a `DataPropertyRange` axiom, the second argument is a data range rather than a class expression.

Object properties can also be declared to be functional, inverse functional, transitive, (ir)reflexive, and (a)symmetric, while data properties can be declared functional. Using `InverseObjectProperty`, one can specify an object property to be the inverse of another. This is not possible with data properties.

Example 23.

```
FunctionalObjectProperty(ex:hasMother)
InverseObjectProperty(ex:hasMother ex:motherOf)
```

A.2.1.11. Assertions about Individuals

Regarding individuals, one can assert: that an individual is a member of a class (`ClassAssertion`); that an individual is related to another via an object property (`ObjectPropertyAssertion`) or to a literal via a data property (`DataPropertyAssertion`); that an individual is *not* related to an individual or literal by a given property (`NegativeObjectPropertyAssertion` and `NegativeDataPropertyAssertion`, respectively); and that two or more individuals are the same (`SameIndividual`) or different (`DifferentIndividuals`).

Example 24.

```
ClassAssertion(ex:FlyingThing ex:superman)
SameIndividual(ex:clark ex:superman)
DifferentIndividuals(ex:clark ex:luther ex:lois)
ObjectPropertyAssertion(ex:friendOf ex:lois ex:clark)
NegativeObjectPropertyAssertion(ex:friendOf ex:luther ex:superman)
NegativeDataPropertyAssertion(ex:name ex:clark "Lex"^^xsd:string)
```

The existence of `SameIndividual` indicates that a single individual in an OWL ontology can be named by multiple IRIs. Like RDF, OWL does not make the unique name assumption.

A.2.1.12. Key Axioms

A key axiom `HasKey(ClassExpr([OPE Expr]*)([DPE Expr]*))` asserts that each named individual belonging to a given class expression is uniquely identified by the values of the specified object and data property expressions (one or both of the property lists must be nonempty). Since OWL rejects the unique name assumption, named individuals sharing the same property values are inferred to be the same individual.

A.3. OWL 2 Semantics

OWL is based on a class of logics called *description logics* (DLs) [4], which are decidable fragments of first order logic. OWL 2 is specifically based on the description logic *SROIQ* [17], and most of the structural components of OWL 2

Table A.7. OWL 2 Ontology and Axiom Structure

<i>OWL ontology</i>	<i>syntax</i>
Ontology Document	[Prefix(<i>Name</i> = <i>IRI</i>)]* <i>Ontology</i>
Ontology	<i>Ontology</i> ([<i>OntIRI</i> [<i>VerIRI</i>]] [Import(<i>IRI</i>)]* [<i>Annotation</i>]* [<i>Axiom</i>]*)
<i>class axioms</i>	<i>syntax</i>
Subclass	SubClassOf(<i>ClassExpr ClassExpr</i>)
Equivalent	EquivalentClasses(<i>ClassExpr</i> [<i>ClassExpr</i>]+)
Disjoint	DisjointClasses(<i>ClassExpr</i> [<i>ClassExpr</i>]+)
Disjoint Union	DisjointUnion(<i>IRI ClassExpr</i> [<i>ClassExpr</i>]+)
<i>object property axioms</i>	<i>syntax</i>
Subproperties	SubObjectPropertyOf(<i>subOPE Expr OPE Expr</i>)
Equivalent Properties	EquivalentObjectProperties(<i>OPE Expr</i> [<i>OPE Expr</i>]+)
Disjoint Properties	DisjointObjectProperties(<i>OPE Expr</i> [<i>OPE Expr</i>]+)
Inverse Properties	InverseObjectProperties(<i>OPE Expr OPE Expr</i>)
Property Domain	ObjectPropertyDomain(<i>OPE Expr ClassExpr</i>)
Property Range	ObjectPropertyRange(<i>OPE Expr ClassExpr</i>)
Functional Property	FunctionalObjectProperty(<i>OPE Expr</i>)
Inverse Functional Property	InverseFunctionalObjectProperty(<i>OPE Expr</i>)
Reflexive Property	ReflexiveObjectProperty(<i>OPE Expr</i>)
Irreflexive Property	IrreflexiveObjectProperty(<i>OPE Expr</i>)
Symmetric Property	SymmetricObjectProperty(<i>OPE Expr</i>)
Asymmetric Property	AsymmetricObjectProperty(<i>OPE Expr</i>)
Transitive Property	TransitiveObjectProperty(<i>OPE Expr</i>)
<i>data property axioms</i>	<i>syntax</i>
Subproperties	SubDataPropertyOf(<i>DP DP</i>)
Equivalent Properties	EquivalentDataProperties(<i>DP</i> [<i>DP</i>]+)
Disjoint Properties	DisjointDataProperties(<i>DP</i> [<i>DP</i>]+)
Property Domain	DataPropertyDomain(<i>DP ClassExpr</i>)
Property Range	DataPropertyRange(<i>DP DR</i>)
Functional Property	FunctionalDataProperty(<i>DP</i>)
<i>assertions</i>	<i>syntax</i>
Same Individual	SameIndividual(<i>Ind</i> [<i>Ind</i>]+)
Different Individual	DifferentIndividuals(<i>Ind</i> [<i>Ind</i>]+)
Class	ClassAssertion(<i>ClassExpr Ind</i>)
Object Property	ObjectPropertyAssertion(<i>OPE Expr Ind Ind</i>)
Negative Object Property	NegativeObjectPropertyAssertion(<i>OPE Expr Ind Ind</i>)
Data Property	DataPropertyAssertion(<i>DP Ind Lit</i>)
Negative Data Property	NegativeDataPropertyAssertion(<i>DP Ind Lit</i>)
<i>entity declarations</i>	<i>syntax</i>
Class	Declaration(Class(<i>IRI</i>))
Datatype	Declaration(Datatype(<i>IRI</i>))
Object Property	Declaration(ObjectProperty(<i>IRI</i>))
Data Property	Declaration(DataProperty(<i>IRI</i>))
Annotation Property	Declaration(AnnotationProperty(<i>IRI</i>))
Individual	Declaration(NamedIndividual(<i>IRI</i>))
Datatype Definition	DatatypeDefinition(<i>IRI DR</i>)
Key Axiom	HasKey(<i>ClassExpr</i> ([<i>OPE Expr</i>]*) ([<i>DPE Expr</i>]*) <i>DR</i>)

Elements enclosed in [...] are optional. [...]*=0 or more. [...]+=1 or more.

ClassExpr=class expression; *Ind*=individual; *OP*=object property; *OPE Expr*=object property expression; *DP*=data property; *DPE Expr*=data property expression; *DR*=data range; *Lit*=literal. *subOPE Expr* has form *OPE Expr*or *ObjectPropertyChain*(*OPE Expr* [*OPE Expr*]+).

Table A.8. OWL 2 Class, Property, and Data Range Expressions

<i>class expressions</i>	<i>syntax</i>
Named Class	<i>IRI</i>
Universal Class	<i>owl:Thing</i>
Empty Class	<i>owl:Nothing</i>
<i>Boolean connectives and enumerated classes</i>	
Class Intersection	<i>ObjectIntersectionOf(ClassExp [ClassExp]+)</i>
Class Union	<i>ObjectUnionOf(ClassExp [ClassExp]+)</i>
Class Complementation	<i>ObjectComplementOf(ClassExp)</i>
Enumerated Class	<i>ObjectOneOf([ClassExp]+)</i>
<i>object property restrictions</i>	
Universal Quantification	<i>ObjectAllValuesFrom(OPExpr ClassExp)</i>
Existential Quantification	<i>ObjectSomeValuesFrom(OPExpr ClassExp)</i>
Individual Value	<i>ObjectHasValue(OPExpr Ind)</i>
Self-Restriction	<i>ObjectHasSelf(OPExpr)</i>
<i>object property cardinality restrictions</i>	
Minimum Cardinality	<i>ObjectMinCardinality(n OPExpr [ClassExp])</i>
Maximum Cardinality	<i>ObjectMaxCardinality(n OPExpr [ClassExp])</i>
Exact Cardinality	<i>ObjectExactCardinality(n OPExpr [ClassExp])</i>
<i>data property restrictions</i>	
Universal Quantification	<i>DataAllValuesFrom([DP]+ DR)</i>
Existential Quantification	<i>DataSomeValuesFrom([DP]+ DR)</i>
Literal Value	<i>DataHasValue(DPLiteral)</i>
<i>data property cardinality restrictions</i>	
Minimum Cardinality	<i>DataMinCardinality(n DP [DR])</i>
Maximum Cardinality	<i>DataMaxCardinality(n DP [DR])</i>
Exact Cardinality	<i>DataExactCardinality(n DP [DR])</i>
<i>object property expressions</i>	
Named Object Property	<i>IRI</i>
Top Object Property	<i>owl:topObjectProperty</i>
Bottom Object Property	<i>owl:bottomObjectProperty</i>
Inverse Object Property	<i>ObjectInverseOf(IRI)</i>
<i>data property expressions</i>	
Named Data Property	<i>IRI</i>
Top Data Property	<i>owl:topDataProperty</i>
Bottom Data Property	<i>owl:bottomDataProperty</i>
<i>data range expressions</i>	
Datatype	<i>IRI</i>
Data Range Intersection	<i>DataIntersectionOf(DR [DR]+)</i>
Data Range Union	<i>DataUnionOf(DR [DR]+)</i>
Data Range Complementation	<i>DataComplementOf(DR)</i>
Literal Enumeration	<i>DataOneOf([Lit]+)</i>
Datatype Restriction	<i>DatatypeRestriction(IRI [IRI Lit]+)</i>

Elements enclosed in [...] are optional. [...]*=0 or more. [...]+=1 or more.

ClassExpr=class expression; *Ind*=individual; *OP*=object property; *OPExpr*=object property expression; *DP*=data property; *DPExpr*=data property expression; *DR*=data range; *Lit*=literal.

discussed so far have direct counterparts in \mathcal{SROIQ} . Upon this view, an OWL ontology amounts to a collection of description logic axioms. However, OWL is also designed to be compatible with RDF. OWL ontologies can be expressed using the RDF/XML concrete syntax. As such, they can be viewed as RDF graphs.

Based upon these views, two distinct varieties of OWL 2 have been defined. OWL 2 DL places certain syntactic restrictions [23] on ontologies and interprets ontologies under a model theoretic *Direct Semantics* [22] based on that of \mathcal{SROIQ} . In OWL 2 Full, ontologies are interpreted according to an *RDF-Based Semantics* [27]. Each provides its own account of satisfaction and entailment.

The two semantics are not equivalent. Like \mathcal{SROIQ} , reasoning in OWL 2 DL is decidable (the syntactic restrictions placed on OWL 2 DL ontologies are put in place to maintain decidability), while reasoning in OWL 2 Full is not. The two semantics are, however, related. Given certain restrictions, OWL 2 DL can be viewed as sound with regard to the RDF-based semantics. The precise nature of the correspondence is given in [27].

We will not discuss either semantics in detail. However, Table A.9 presents the syntax and semantics for structures common to description logics, and this provides insight into the OWL 2 DL semantics. We also note that the DL semantics also provides semantics for the decidable subsets of OWL 1 and for the tractable OWL profiles described below. Reasoning in each of these is decidable.

A.3.1. OWL 2 Profiles

For a given ontology, there are a number of reasoning tasks commonly performed.

- *consistency checking*: Checking whether an ontology has any models.
- *instance checking*: Checking whether individual a belongs to a class C .
- *subsumption checking*: Checking whether class C is a subclass of D .
- *class satisfiability checking*: Checking whether a class can have instances.

These tasks are defined relative to an ontology. There are other tasks (in *classification*, the entire class subsumption hierarchy is computed), and in general an instance of one task can be readily converted into an instance of another (and they can all be viewed as entailment problems). E.g., C is unsatisfiable in an ontology O if and only if it is a subclass of \perp (that is, $O \models C \sqsubseteq \perp$).

The problems noted above are decidable in OWL 2 DL, but they have a high worst-case complexity (they are N2ExpTime-complete). This has led to the development of three syntactic fragments of OWL 2 called *profiles* [20]. The profiles—OWL 2 EL, OWL 2 QL, and OWL 2 RL—are designed with specific applications in mind and are of a lower computational complexity than OWL 2 DL. In each case, the complexity results are achieved by eliminating features—such as negation or disjunction—from the logics. Grammars for each logic are presented in the W3C OWL 2 Profiles specification [20].

A.3.1.1. OWL 2 EL

OWL 2 EL is based on the description logic \mathcal{EL}^{++} [2, 3], which has been used in the construction of large biomedical ontologies (in practice, it is possible to quickly

Table A.9. Description logic syntax and semantics

<i>expression type</i>	<i>syntax</i>	<i>semantics</i>
named class	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
named individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
property	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse property	R^{-}	$R^{-\mathcal{I}} = \{(y, x) (x, y) \in R^{\mathcal{I}}\}$
enumeration	$\{a_1, \dots, a_n\}$	$\{a_1^{\mathcal{I}}, \dots, a_n^{\mathcal{I}}\}$
<code>owl:Thing</code>	\top	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
<code>owl:Nothing</code>	\perp	$\perp^{\mathcal{I}} = \emptyset$
complementation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} / C^{\mathcal{I}}$
intersection	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
union	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
exists	$\exists R.C$	$\{x (\exists y)[(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}]\}$
value	$\forall R.C$	$\{x (\forall y)[((x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$
self	$\exists R.\text{Self}$	$\{x (x, x) \in R^{\mathcal{I}}\}$
atmost	$\leq n R.C$	$\{x \#\{y (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$
atleast	$\geq n R.C$	$\{x \#\{y (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$
<i>axiom/assertion</i>	<i>syntax</i>	<i>semantics</i>
subclass	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
subproperty	$R_1 \circ \dots \circ R_n \sqsubseteq R_{n+1}$	$(R_1 \circ \dots \circ R_n)^{\mathcal{I}} \subseteq R_{n+1}^{\mathcal{I}}$
class assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
individual equality	$a_i = a_j$	$a_i^{\mathcal{I}} = a_j^{\mathcal{I}}$
individual inequality	$a_i \neq a_j$	$a_i^{\mathcal{I}} \neq a_j^{\mathcal{I}}$
property assertion	$R(a_i, a_j)$	$(a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \in R^{\mathcal{I}}$
negative property assertion	$\neg R(a_i, a_j)$	$(a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \notin R^{\mathcal{I}}$

Here, a , C , and R stand for named individuals, class expressions, and property expressions, respectively. An interpretation \mathcal{I} maps these to structures defined over a nonempty domain $\Delta^{\mathcal{I}}$. An interpretation \mathcal{I} satisfies (is a model of) an axiom if the corresponding condition above is met. It is a model of an ontology O if it is a model of its axioms. O entails an axiom A ($O \models A$) iff every model of O is a model of A .

compute the class hierarchies of ontologies containing many thousands of classes). The desirable properties of OWL 2 EL are obtained by restricting class expressions to conjunction and existential quantification. `IntersectionOf` and `SomeValuesFrom` are allowed (as are `HasValue` and `HasSelf`), but `UnionOf`, `ComplementOf`, and `AllValuesFrom`, are not. Cardinality restrictions are similarly disallowed. Enumerations can be used, provided they have only one element.

Only these restricted class expressions can be used in subclass, equivalent class, and disjoint class axioms. Inverse properties are not allowed in any expression, nor are anonymous individuals.

The following other types of axiom are allowed in OWL 2 EL (again with restricted class expressions): equivalent property, disjoint property, and subproperty axioms (including those with property chains); reflexive and transitive property axioms; domain and range axioms; key axioms; class, property, and negative property assertions; and individual equality and inequality assertions.

A.3.1.2. OWL 2 QL

OWL 2 QL is based on the DL-Lite [1] family of description logics and is designed for query answering over large ABoxes. Syntactically, a distinction is made between the lefthand and righthand side of subclass axioms $B \sqsubseteq C$, i.e. between *subclass* (B) and *superclass* (C) expressions. Using description logic notation, each can have the form below.

$$\begin{aligned} B ::= & \quad A \mid \exists R. \top \\ C ::= & \quad A \mid C \sqcap C \mid \neg B \mid \exists R. A \end{aligned}$$

Here, A is a class name (including \top and \perp) and R is any OWL 2 property expression. In addition to subclass axioms, equivalent and disjoint class axioms are allowed provided that the class expressions involved are subclass expressions.

Property axioms other than transitive and functional property axioms can be used, except that property chains are not allowed. Also, only superclass expressions can be used in domain and range axioms. Class assertions (where the class expression is a name), positive property assertions and individual inequality assertions are allowed, but equality assertions are not, nor are negative property assertions or key axioms.

A.3.1.3. OWL 2 RL

OWL 2 RL is designed to allow implementation in rule-based systems, and so its structures are those that permit translation into a set of rules.⁶ As in OWL 2 QL, subclass axioms $B \sqsubseteq C$ distinguish the sub- and superclass expressions.

$$\begin{aligned} B ::= & \quad A \mid \perp \mid B \sqcap B \mid B \sqcup B \mid \{a_1 \dots a_n\} \mid \exists R.B \mid \exists R. \top \\ C ::= & \quad A \mid \perp \mid C \sqcap C \mid \neg B \mid \exists R.\{a\} \mid \forall R.C \mid \leq i R.B \mid \leq i R. \top \\ i ::= & \quad 0 \mid 1 \end{aligned}$$

Above, A cannot be \top . Equivalent and disjoint class axioms are allowed, and in the latter, class expressions must be subclass expressions. In equivalent class axioms, class expressions must have the form

$$D ::= A \mid \perp \mid D \sqcap D \mid \exists R.\{a\}.$$

Property axioms are the same as those in OWL 2 DL, except that properties cannot be declared reflexive, and only superclass expressions can be used in domain axioms, range axioms, and positive class and property assertions. Individual assertions are otherwise as they are in OWL 2.

A.3.1.4. Reasoning in the Profiles

For each of the OWL 2 profiles, the reasoning tasks noted in Section A.3.1 can be performed in polynomial time relative to the total size of the ontology (and input expressions involved, if any). They apply to OWL 2 EL and QL. They also apply to OWL 2 RL provided the class expressions involved in class satisfiability, subsumption checking, and instance checking are atomic; otherwise, these tasks become co-NP-complete. A more detailed discussion of the results can be found in the W3C OWL 2 Profiles specification [20].

⁶A set of rules, similar to those for RDFS, is presented in the OWL 2 RL standard [20].

Bibliography

- [1] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* Family and Relations. *Journal of Artificial Intelligence Research*, 36(1):1–69, Sept. 2009.
- [2] F. Baader, S. Brandt, and C. Lutz. Pushing the \mathcal{EL} Envelope. In *Proceedings of the 19th International Joint Conference on Artificial intelligence (IJCAI '05)*, pages 364–369. Morgan Kaufmann Publishers Inc., 2005.
- [3] F. Baader, S. Brandt, and C. Lutz. Pushing the \mathcal{EL} Envelope Further. In K. Clark and P. F. Patel-Schneider, editors, *Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.
- [4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, New York, NY, USA, 2007.
- [5] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. January 2005. RFC 3986. Available at <https://www.ietf.org/rfc/rfc3986>, 2005.
- [6] T. Bray, D. Hollander, A. Layman, R. Tobin, and H. S. Thompson, editors. Namespaces in XML 1.0 (Third Edition). W3C Recommendation, 8 December 2009. Available at <http://www.w3.org/TR/xml-names/>, 2009.
- [7] T. Bray, D. Hollander, A. Layman, and R. Tobin, editors. Namespaces in XML 1.1 (Second Edition). W3C Recommendation, 16 August 2006. Available at <https://www.w3.org/TR/xml-names11/>, 2006.
- [8] D. Brickley and R. Guha, editors. RDF Schema 1.1. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/rdf-schema/>, 2014.
- [9] G. Carothers and A. Seaborne, editors. RDF 1.1 N-Triples: A line-based syntax for an RDF graph. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/n-triples/>, 2014.
- [10] G. Carothers and A. Seaborne, editors. RDF 1.1 TriG: RDF Dataset Language. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/trig/>, 2014.
- [11] G. Carothers, editor. RDF 1.1 N-Quads: A line-based syntax for RDF datasets. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/n-quads/>, 2014.
- [12] R. Cyganiak, D. Wood, and M. Lanthaler, editors. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/rdf11-concepts/>, 2014.
- [13] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). January 2005. RFC 3987. Available at <http://www.ietf.org/rfc/rfc3987>, 2005.
- [14] F. Gandon and G. Schreiber, editors. RDF 1.1 XML Syntax. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/rdf-syntax-grammar/>, 2014.
- [15] P. J. Hayes and P. F. Patel-Schneider, editors. RDF 1.1 Semantics. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>, 2014.

- [16] M. Horridge and P. F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition). W3C Working Group Note 11 December 2012. Available at <http://www.w3.org/TR/owl2-manchester-syntax/>, 2012.
- [17] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible *SROIQ*. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.
- [18] G. Klyne and J. J. Carroll, editors. Resource Description Framework (RDF): Concepts and Abstract Syntax W3C Recommendation, 10 February 2004. Available at <https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, 2004.
- [19] D. L. McGuinness and F. van Harmelen, editors. OWL Web Ontology Language Overview. W3C Recommendation, 10 February 2004. Available at <http://www.w3.org/TR/owl-features/>, 2004.
- [20] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, editors. OWL 2 Web Ontology Language Profiles (Second Edition). W3C Recommendation 11 December 2012. Available at <http://www.w3.org/TR/owl2-profiles/>, 2012.
- [21] B. Motik, B. Parsia, and P. F. Patel-Schneider, editors. OWL 2 Web Ontology Language: XML Serialization (Second Edition). W3C Recommendation, 11 December 2012. Available at <http://www.w3.org/TR/owl2-xml-serialization/>, 2012.
- [22] B. Motik, P. F. Patel-Schneider, and B. Cuenca Grau, editors. OWL 2 Web Ontology Language: Direct Semantics (Second Edition). W3C Recommendation, 11 December 2012. Available at <http://www.w3.org/TR/owl-direct-semantics>, 2012.
- [23] B. Motik, P. F. Patel-Schneider, and B. Parsia, editors. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation, 11 December 2012. Available at <https://www.w3.org/TR/owl2-syntax/>, 2012.
- [24] D. Peterson, S. Gao, A. Malhotra, C. M. Sperberg-McQueen, and H. S. Thompson, editors. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Recommendation, 5 April 2012. Available at <http://www.w3.org/TR/xmldschema11-2/>, 2012.
- [25] A. Phillips and M. Davis, editors. Tags for Identifying Languages. September 2009. RFC 5646. Available at <https://tools.ietf.org/html/rfc5646>, 2009.
- [26] E. Prud'hommeaux and G. Carothers, editors. RDF 1.1 Turtle: Terse RDF Triple Language. W3C Recommendation, 25 February 2014. Available at <https://www.w3.org/TR/turtle/>.
- [27] M. Schneider, editor. OWL 2 Web Ontology Language: RDF-Based Semantics (Second Edition). W3C Recommendation, 11 December 2012. Available at <http://www.w3.org/TR/owl2-rdf-based-semantics/>, 2012.
- [28] M. Sporny, G. Kellogg, and M. Lanthaler, editors. JSON-LD 1.0: A JSON-based Serialization for Linked Data. W3C Recommendation, 16 February 2014. Available at <http://www.w3.org/TR/json-ld/>, 2014.

- [29] H. J. ter Horst. Extending the RDFS Entailment Lemma. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *LNCS*, pages 77–91. Springer, 2004.
- [30] H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.
- [31] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation, 11 December 2012. Available at <http://www.w3.org/TR/owl-overview>, 2012.
- [32] W3C SPARQL Working Group. SPARQL 1.1 Overview. W3C recommendation, 21 March 2013. Available at <http://www.w3.org/tr/sparql11-overview/>, 2013.

This page intentionally left blank

Index

- Protégé, *see* ontology editor
- 303 URI, *see* Uniform Resource Identifier, 303
- 3D, 107
- 3D+1, 107, 112
- 4D, 107, 114
- abstraction level, 65
- agent, 315
- agent role ODP, 7, 12
- Agent Role pattern, 271, 316
 - AgentRole class, 317
- agile ontology engineering, 26
- agri-food, 289
- alignment, 25, 40, 43
- Alloy, 167
- Analytic Hierarchy Process, 55
- anti-pattern, 164, 166, 172, 177, 184, 185, 190, 258
- architecture, xi
- Aristotle, 73
- Artificial Intelligence, 73
- Association of Ontology Design and Patterns, 81
- ATLAS, 329
- autoepistemic logic, 74
- axiom, 12
- axiomatization, 12, 73, 140, 146, 191
- backlog, 32
- barcode, 280
- best-practice artifact, 259
- BFO, 193
- Bibliographic Framework, 309
- Boolean, 332
- building block, xi
- case-based reasoning, 125
- CERN, 329
- chess, 3
- CIDOC, 310
- CIDOC-CRM, 193
- circumscription, 74
- class diagram, 78, 95
- class in-degree, 63
- class out-degree, 63
- class satisfiability, 356
- class-oriented specialisation, 25
- classification, 89, 356
- classification scheme, 247
- classification time, 108
- closed world assumption, 74
- CMS, 44, 329
- co-localisation, 84
- co-presence, 84
- code smell, 164
- CODP Quality Model, 51, 56
- collection, 90
- Collection and Membership ODP, 90
- Collection ontology, 82, 86
- common language, 41
- compatibility, 60, 66
- competency question, 4, 34, 37, 140, 146, 330
- component, 91
- component diagram, 94
- Component ODP, 91
- composition, 25, 191
- computational linguistics, 193
- concepts as individuals, 120
- conceptual frame, 299
- conceptual model, 165
- conceptualization, 106
- consistency, 356
- constituent, 91

- constitution, 84
 Constitution ODP, 91
 constraint, 191
 content, 299
 content negotiation, 211, 215
 Content ODP, 24, 270
 content ontology design pattern, 134
 context, 120
 context indexing, 115
 Contextual Statement, 34, 37
 control checking, 121
 control flow, 136
 copy, 299
 copyright, 305
 core ontology, 133, 140, 142
 coreference resolution, 227
 creative work, 299
 criminal profiling, 45
 Cruise pattern, 271
 cultural heritage, 310
 customer, 26, 47
 cyclomatic complexity, 63
- D&S, 82, 84
 data, 75
 data integration, 267
 architecture, 269
 challenges, 267
 extendible, 267
 flexible, 267
 ontology-based, 269
 ontology-based linked data, 270
- Datalog reasoner, 223
 datatype, 340
 DBpedia, 227
 de re vs. de dicto, 299
 debugging, 43
 default logic, 74
 dereferenceable, *see* Web-dereferenceable
 description, 92, 120–122
 Description and Situation ontology, 82, 84
 description logic, 74, 353, 357
 semantics, 357
 syntax, 357
 Description ODP, 92
- Descriptions and Situations, 120, 121, 123
 Descriptive Ontology for Linguistic and Cognitive Engineering, 82
 design pair, 33
 design pattern, xi, 105
 detector final state, 329
 diagnosis, 121, 123
 direct reuse, 82
 disambiguation, 76
 disjointness, 14
 distributed development, 28, 46
 DL reasoning, 108
 document engineering, 299, 306
 documentation, 190
 documentation indicators, 60
 documentation minimalism, 62
 DOLCE, 82, 121, 305
 DOLCE+D&S Ultralite, 86, 97
 Dolce+D&S Ultralite, 81, 99
 Dolce-D&S Ultralite, 81
 DOLCE-Lite, 86
 domain, 13, 78
 domain expert, 27
 domain restriction
 scoped, 13, 17
 domain-related ontology pattern, 133–135, 142, 146, 157
 dual schema, 204, *see* schema, dual populating, 217
 duality, 125
 Dublin Core, 308
 DUL, 81, 86, 97, 99
 duper description, 120, 123
- EARMARK, 306
 EEM, 281
 electronic pedigree, 280
 endurant, 83, 86
 Entity Classification ODP, 89
 entity-relationship model, 53, 64
 EPC, 281
 EPCIS, 280
 evaluation, 190
 event, 83, 86, 87, 315
 event ODP, 8, 14

- Event pattern, 271
event type, 105
Events and participation ODP, 87
existential axiom, 87
expansion, 25
export interface, 94
expression, 300, 305, 306, 308
expressivity, 116, 117, 121
extensional multigrade predicate, 106,
 107
eXtreme Design, 51, 66, 191
eXtreme Design (XD), 23, 26

fishery, 44
fix, 321, 323
fluent, 107
foundational ontology, 82, 84, 142
frame logic, 112, 121
frames, 122
framing, 120
FRBR, 308
FreeRole, 175
functional suitability, 58, 66

generalisation, 25, 191
GeoLink, 267
GeoSPARQL, 322
global schema, *see* schema, global
goal, 93
GUIPOT interface, 257

Hash URI, *see* Uniform Resource
 Identifier, hash
hasKey axiom, 112
heterogeneity, xiii
HOL, 121
holds at, 109
HTTP URI, *see* Uniform Resource
 Identifier, HTTP
hypothesis testing, 125

IDE, 194
import, 25
import interface, 94
in-use indicators, 60
incomplete information, 125
inconsistency, 77
individual indexing, 107, 114, 115

inference, 76
inferential semantics, 75
information entity, 299, 300, 302, 305
information object, 92, 303, 305
Information Object ontology, 82, 85
Information Realisation ODP, 92
information realization, xii, 92, 299,
 303, 305, 309, 310
information system, 52
inherence, 84
instance checking, 356
instance-free sentence, 33
integration, 40
integration testing, 41
integrity constraint, 75, 77
intensional multigrade predicate, 106,
 120
intensional reification, 122
Internationalized Resource Identifier,
 338–339
 absolute, 339
 dereferencing, 339
 relative, 339
interoperability, 24, 119, 234, 311
invariant, xi
IRI, *see* Internationalized Resource
 Identifier
ISO 25010, 52, 58
iterative development, 27

jet, 330

knowledge acquisition, 233
knowledge encapsulation, 94
knowledge gaps, 65
knowledge pattern, xi, 105, 113
knowledge pattern science, 105
knowledge representation, 73

language downgrading, 258
language profiling, 258
legal knowledge, 123
lepton, 330
lessons learned, 43
lexicon, 247
LHCb, 329
library and information science, 308
linguist resources, 311

- linguistic act, 300, 303
 linguistic knowledge, 299
 linked data, 4
 Linked Data, xiii, 119, 201, 235, 239, 268
 front-end, 228
 publishing, 202
 linked data pattern, xi
 linked dataset, 202, 207
 Linked Open Data, 201
 Linked Open Data Cloud, 227
 linked pedigree, 280
 LinkedEPCIS, 287
 literal
 in OWL 2, 349, 351
 in RDF, 338, 339
 in Turtle, 341, 342
 local closed world, 75
 location, 91
 Location ODP, 91
 LODE, 216
 logic, 73
 logic programming, 75
 logical expressivity, 108
 logical ontology design pattern, 134
 maintainability, 59, 66
 manifestation, 300, 305, 306, 308
 markup semantics, 306
 matching ODP, 36
 meaning, 122, 300
 media, 305
 membership, 90
 methodology, 26
 model indicators, 60
 model validation, 167, 184
 modeling, 271
 collaborative, 271
 Content Pattern, 271
 pattern, 271
 modelling issue, 87
 modelling solution, 87
 modular ontology, 27, 270
 modular schema, *see* schema, modular
 modularization, 195
 module, 11, 270
 module composition, 95
 monotonicity, 74
 MultDep, 177, 178
 multidimensional design space, 117
 multigrade predicate, 105
 multimedia semantics, 311
 multiple interpretations, 120
 n-ary relationship, 105
 name nesting, 116
 named graph, 109
 namespace, 95
 naming convention, 191
 natural language processing, 193
 NeOn Toolkit, *see* ontology editor
 nominal, 111
 non-functional requirements, 31
 non-monotonicity, 74
 NOR2OWL transformation, 246, 247
 normative interpretation, 121, 125
 O^2 , 55
 object, 83, 86
 OBM2OWL transformation, 246, 248
 OBO ontology, 248
 ocean science, 268
 oceanographic cruise, 271
 oceanography, *see* ocean science
 OCL, 172
 ODM, 94
 ODP, 81, 87, 99
 ODP composition, 36
 ODP construction, 36
 ODP development, 28
 ODP extension, 36
 ODP import, 36
 ODP minimalism, 62
 ODP operations, 25
 ODP repository, 43
 ODP search engine, 43
 ODP size, 63
 ODP specialisation, 36, 43
 ODP specialisation strategy, 63
 ODP tooling, 68
 ODP-based design, 95
 ODPA, 68, 81
 OntoClean, 56, 64
 ontological analysis, 142

- ontological anti-pattern, 161, 165, 167, 184
ontological background model, 246
ontological commitment, 83
ontology, 207, 348
 ABox, 348
 as global schema, 270
 populating, 216
 TBox, 348
ontology alignment, 195, 236
ontology coding pattern, 134
Ontology Definition Metamodel, 94
ontology design pattern, xi, 81, 82, 87, 99
ontology design pattern composition, 95, 97
ontology development process, 134
ontology editor, 214
ontology engineering, 233
ontology engineering methodology, 191
ontology engineering tool, 42
ontology engineering tradeoffs, 66
ontology evolution, 245
ontology interface, 94
ontology matching, 262
ontology modularisation, 94
ontology module, 94
ontology pattern, 134
ontology pattern language, 133–135, 140, 142, 147, 157
ontology pitfalls, 166
ontology population, 196
ontology project, 96
ontology quality, 45, 54
ontology transformation, 245
ontology-based data integration, 269
ontology-based linked data integration, 270
ontology-driven conceptual modeling, 161, 163, 165, 184
OntologyDesignPatterns.org, 68, 314
OntologyDesignnPtterns.org, 81
ONTOMETRIC, 55
OntoPedigree, 281
OntoUML, 161–163, 165, 167, 169, 248
OO design patterns, 54
open world assumption, 74
OPPL, 43, 251
oQual, 55
OWL, 12, 73, 191, 348–358
OWL 1, 348
OWL 2, 348–358
 anonymous individual, 349
 assertion, 353
 axiom, 354
 cardinality restriction, 350
 class axiom, 352
 class constructor, 349
 class expression, 349, 355
 constraining facet, 351
 data property, 349
 data property expression, 351, 355
 data range, 351, 355
 datatype definition, 351
 individual value restriction, 351
 key axiom, 353
 named individual, 349
 object property, 349
 object property expression, 351, 355
 OWL 2 DL, 356
 OWL 2 EL, 356–357
 OWL 2 Full, 356
 OWL 2 QL, 356, 358
 OWL 2 RL, 356, 358
 property axiom, 352
 property chain, 352
 self restriction, 351
OWL 2 Functional Syntax, 349–353
 ClassAssertion, 353
 DataComplementOf, 351
 DataIntersectionOf, 351
 DataOneOf, 351
 DataPropertyAssertion, 353
 DataPropertyRange, 353
 DataUnionOf, 351
 DatatypeDefinition, 352
 DatatypeRestriction, 352
 DifferentIndividuals, 353
 DisjointUnion, 352
 EquivalentClasses, 352

FunctionalObjectProperty, 353
HasKey, 353
InverseObjectProperty, 353
NegativeDataPropertyAssertion, 353
NegativeObjectPropertyAssertion, 353
ObjectAllValuesFrom, 350
ObjectComplementOf, 349
ObjectExactCardinality, 350
ObjectHasSelf, 351
ObjectHasValue, 351
ObjectIntersectionOf, 349
ObjectMaxCardinality, 350
ObjectMinCardinality, 350
ObjectOneOf, 349
ObjectPropertyAssertion, 353
ObjectSomeValuesFrom, 350
ObjectUnionOf, 349
SameIndividual, 353
SubClassOf, 352
SubObjectPropertyOf, 352
owl:Nothing, 349
owl:Thing, 349
owl:bottomDataProperty, 351
owl:bottomObjectProperty, 351
owl:topDataProperty, 351
owl:topObjectProperty, 351
OWL 2 profile, 66, 68
OWL 2 property chain, 126
OWL 2 punning, 121, 126
OWL 2 Semantics, 353–356
OWL anti-pattern, 166
OWL API, 214
OWL axiom annotation, 110
OWL building block, 24
OWL imports, 63
OWL2OWL transformation, 246, 250
pair design, 29
parameter, 89
Parameter Classification ODP, 89
part of, 91
part-whole, 91
Part-whole ODP, 91
parthood, 84
Participant Role ODP, 36
participation, 83, 84, 87, 94
particle, 330
particle detector, 329
Particle Physics, 329
Patomat, 43
PatOMat project, 252
pattern action, 135
pattern language, 134, 140
patterns group, 135, 138
perdurant, 83, 86
perishable goods, 289
pharmaceutical, 289
physical object, 305
place, 239
plan, 93
plan model, 121
Plan ODP, 93
Plan ontology, 82, 85
planning, 123
polymorphism, 108, 109, 111, 113–116, 118
populating ontology, *see* ontology, populating
project ODP catalogue, 24
project roles, 46
property-oriented specialisation, 25
Protégé, 67
publication, 330
punning, 84
PURL, 213
PURO modeling, 249
quadruple, 109
quale, 83, 84
quality, 83, 88, 190
quality characteristic, 57
quality indicator, 57
Quality of entities, 88
query time, 108
quintuple, 110
range, 13, 78
range restriction
 scoped, 13, 17
rapid prototyping, 27
RDF, 73, 337–347

- blank node, 340, 342
collection, 343, 344
concrete syntax, 338, 341
dataset, 340
document, 341
graph, 337
graph isomorphism, 340
reification, 344, 345
statement, 337, 338
term, 338
triple, 337
RDF API, 218
RDF reification, 110
RDF Semantics, 346–348
 axiomatic triple, 347
 datatype entailment, 347
 entailment rule, 348
 RDF entailment, 347
 RDFS entailment, 347
 simple entailment, 346
 simple interpretation, 346
RDF Syntax, 337
 `rdf:Alt`, 344
 `rdf:Bag`, 344
 `rdfs:Class`, 344
 `rdfs:comment`, 345
 `rdfs:Container`, 344
 `rdfs:Datatype`, 344
 `rdfs:domain`, 345
 `rdf:first`, 345
 `rdf:HTML`, 344
 `rdfs:isDefinedBy`, 345
 `rdfs:label`, 345
 `rdf:langString`, 344
 `rdfs:Literal`, 344
 `rdfs:member`, 345
 `rdfs:ContainerMembershipProperty`, 344
 `rdf:_n`, 345
 `rdf:object`, 345
 `rdf:predicate`, 345
 `rdf:Property`, 344
 `rdfs:range`, 345
 `rdfs:Resource`, 344
 `rdf:rest`, 345
 `rdfs:seeAlso`, 345
 `rdf:Seq`, 344
 `rdf:Statement`, 344
 `rdfs:subClassOf`, 343
 `rdfs:subClassOf`, 345
 `rdf:subject`, 345
 `rdfs:subPropertyOf`, 345
 `rdf:type`, 345
 `rdf:value`, 345
 `rdf:XMLLiteral`, 344
RDFS, 343–345
re-engineering pattern, 247
reasoning, 76
 reasoning performance, 60
Reasoning Requirement, 34, 37
refactoring, 40
reference, 300
reference domain ontology, 134, 142
region, 83, 84, 88
Region ODP, 88
regression testing, 37
relation footprint, 110–116
relation indexing, 107, 112, 115
relation topology, 108
release, 40, 41
RelRig, 172, 175
repository, 194
repurposing, 195
requirements, 105
requirements engineering, 27, 33
research questions, 189
resource, 337
reuse, 23, 28, 30, 47, 97, 134, 142, 146, 190
RFID, 280
RIF, 73, 127
role, 89, 163, 169, 175, 177, 184
 `role`, 313
 time-indexed, 315
Role Classification ODP, 89
Role pattern, 315
 Role class, 316
 `performsRole` property, 316
 `providesRole` property, 316
 `rolePerformedBy` property, 316
 `roleProvidedBy` property, 316
rule, 75, 191
schema, 75

- dual, 204, 268
- global, 269
- modular, 270
- publishing, 214
- scoping, 5, 30
- selection criteria, 332
- semantic heterogeneity, 234
- semantic integration, 311
- semantic trajectory, 321
- semantic web, 166
- semiotics, 300, 303
- sensor, 241
- sentence indexing, 107, 109–111
- sequence, 92
- Sequence ODP, 92
- service, 123
- SHACL, 75
- shape constraint, 78
- shortcut, 191
- singleton domain, 111
- singleton property, 110
- situation, 93, 112, 113, 120, 125, 127, 301
- Situation ODP, 93
- SKOS, 121
- SKOS vocabulary, 252
- small projects, 46
- SNOMED-CT, 193
- social object, 305
- software developer, 26
- software engineering, 45
- SPARQL, 69
 - CONSTRUCT query, 224, 225
 - SPARQL query, 37
- spatial region, 84
- spatiotemporal, 322
- specialisation, 25, 191
- specificity, 190
- SROTQ*, 353
- SSN-XG, 324
- standard, 191
- standardization, 75
- statement abstraction, 111
- statement contextualization, 109
- statement reification, 110
- stereotype, 94
- stub, 10
- subsumption, 356
- subsumption breadth, 63
- subsumption depth, 63
- SUMO, 193
- super-duper description, 126, 127
- supply chain, 279
- SWRL, 76
 - DL-safe, 215
 - reasoner, 223
- synthetic knowledge base, 109
- tagged value, 95
- tangledness, 63
- task, 90
- Task Classification ODP, 90
- task-focused ontology, 27
- tautology, 76, 78
- teaching, 47
- temporal footprint, 108
- temporal indexing, 107
- temporal region, 84
- TemporalExtent**, 315
 - hasTemporalExtent property, 316
- test-focused development, 28
- testability, 66
- testing, 28, 37
- thesaurus, 247
- time indexed participation, 94
- time interval, 88
- Time Interval ODP, 88
- time-indexed participation, 94
- tool, 194
- trace, 281
- traceability, 279
- track, 281
- trajectory, xii, 238, 239, 321
- triple store, 218
- Turtle, 341–343
- UFO, 162, 163, 165
- UML profile, 82, 94, 96
- Uniform Resource Identifier, 207, 338, 303, 211
 - hash, 210
 - HTTP, 207
 - minting, 207
 - namespace, 210
 - naming scheme, 209, 212

opaque, 209
persistence, 209
unique name assumption, 346
universe of concepts, 123
universe of discourse, 123
universe of facts, 123
URI, *see* Uniform Resource Identifier
usability, 59, 118
use, 95
use case, 4
user story, 27, 31
user story template, 31

variant pattern, 136, 151
Venn diagram, 62
Vessel pattern, 271
view, 191
 consumer-centric, 273
 producer-centric, 272
 representing shortcuts, 206
vocabulary, 202
vocabulary camp, 234, 236, 238, 241
VOWL, 62

W3C, 73
w3id, 213
Web-dereferenceable, 207
WebProtégé, 67, 194
white box, 26

XD for WebProtégé, 42
XML, 306

This page intentionally left blank