Ontology-Driven, Guided Visualisation Supporting Explicit and Composable Mappings

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Medieninf. Jan Polowinski
geboren am 3. März 1981
in Gelsenkirchen

Gutachter

Prof. Dr. rer. nat. habil. Uwe Aßmann,
Technische Universität Dresden

Prof. Dr. Ulrich W. Eisenecker,
Universität Leipzig

Tag der Verteidigung: 20. Januar 2017

Dresden, den 19. Oktober 2017

Abstract

Data masses on the World Wide Web can hardly be managed by humans or machines. One option is the formal description and linking of data sources using Semantic Web and Linked Data technologies. Ontologies written in standardised languages foster the sharing and linking of data as they provide a means to formally define concepts and relations between these concepts. A second option is visualisation. The visual representation allows humans to perceive information more directly, using the highly developed visual sense. Relatively few efforts have been made on combining both options, although the formality and rich semantics of ontological data make it an ideal candidate for visualisation. Advanced visualisation design systems support the visualisation of tabular, typically statistical data. However, visualisations of ontological data still have to be created manually, since automated solutions are often limited to generic lists or node-link diagrams. Also, the semantics of ontological data are not exploited for guiding users through visualisation tasks. Finally, once a good visualisation setting has been created, it cannot easily be reused and shared. Trying to tackle these problems, we had to answer how to define composable and shareable mappings from ontological data to visual means and how to guide the visual mapping of ontological data.

We present an approach that allows for the guided visualisation of ontological data, the creation of effective graphics and the reuse of visualisation settings. Instead of generic graphics, we aim at tailor-made graphics, produced using the whole palette of visual means in a flexible, bottom-up approach. It not only allows for visualising ontologies, but uses ontologies to guide users when visualising data and to drive the visualisation process at various places: First, as a rich source of information on data characteristics, second, as a means to formally describe the vocabulary for building abstract graphics, and third, as a knowledge base of facts on visualisation. This is why we call our approach *ontology-driven.* We suggest generating an *Abstract Visual Model* (AVM) to represent and »synthesise« a graphic following a role-based approach, inspired by the one used by J. v. Engelhardt for the analysis of graphics. It consists of graphic objects and relations formalised in the *Visualisation Ontology* (VISO). A *mappings model*, based on the declarative *RDFS/OWL Visualisation Language* (RVL), determines a set of transformations from the domain data to the AVM. RVL allows for composable visual mappings that can be shared and reused across platforms. To guide the user, for example, we discourage the construction of mappings that are suboptimal according to an effectiveness ranking formalised in the fact base and suggest more effective mappings instead. The guidance process is flexible, since it is based on exchangeable rules. VISO, RVL and the AVM are additional contributions of this thesis. Further, we initially analysed the state of the art in visualisation and RDF-presentation comparing 10 approaches by 29 criteria. Our approach is unique because it combines ontology-driven guidance with composable visual mappings. Finally, we compare three prototypes covering the essential parts of our approach to show its feasibility. We show how the mapping process can be supported by tools displaying warning messages for non-optimal visual mappings, e. g., by considering relation characteristics such as »symmetry«. In a constructive evaluation, we challenge both the RVL language and the latest prototype trying to regenerate sketches of graphics we created manually during analysis. We demonstrate how graphics can be varied and complex mappings can be composed from simple ones. Two thirds of the sketches can be almost or completely specified and half of them can be almost or completely implemented.

Acknowledgements

I would like to thank my supervisor, Prof. Uwe Aßmann, for giving me the opportunity to work on my topic, which fascinated me from day one until now. I appreciate this very much. Thank you as well for many helpful comments. I also would like to thank Prof. Ulrich W. Eisenecker, who accepted to be my external reviewer, for taking the time to comment on my work in detail.

Many thanks also to Jendrik Johannes and Katja Siegemund, who carefully reviewed chapters of this work, for their constructive comments. Thank you also to my former colleagues and friends from the software technology group at TU Dresden for commenting on talks I gave there and for helping me out in my research life. Especially, I would like to thank Christoff Bürger, who shared the office with me, for the discussions on graph transformations and language definitions. Thank you also to Christian Wende for introducing me to OWLText.

The VISO ontology is the result of fruitful discussions with my colleague Martin Voigt from the chair for multimedia technology. As noted in the respective chapter, the chapter on the VISO ontology has been built on work previously published and co-authored with Martin. The initial version of the VISO documentation and its bibliographic annotations was set up and improved many times by Fabian Prager. Thank you for your endurance. Many thanks also to Pooran Patel, one of the students I supervised, who successfully built the second (OntoWiki-based) prototype in spite of many changes to the then premature RVL specification. Thank you Anna and Hermann for proofreading on short call.

This work would not have been possible without the patience and trust of many people including my family and especially my wife Anna. Standing by me all the time I worked on this thesis is something that cannot be taken for granted. Thank you for supporting and encouraging me.

Work on this thesis received financial support from the European Social Fund and the Free State of Saxony, contract no. 80937064.

Europa fördert Sachsen.
ESF
Europäischer Sozialfonds

SACHSEN

Publications

This thesis is partially based on the following peer-reviewed publications:

- J. Polowinski. Towards RVL: a declarative language for visualizing RDFS/OWL data. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics (WIMS '13)*, 38:1–38:11. New York, NY, USA, 2013. ACM.
- J. Polowinski and M. Voigt. VISO: A shared, formal knowledge base as a foundation for semi-automatic InfoVis systems. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI WIP '13)*. Paris, France, 2013. ACM.

The following peer-reviewed publications cover work on *faceted browsing*, which is closely related to the content of the thesis, but not contained herein (we suggest the mechanism of faceted browsing to be used for filtering and selection tasks in our approach):

- J. Polowinski. Widgets for faceted browsing. In *Human Interface and the Management of Information. Designing Information Environments*, LNCS 5617 proceedings, pages 601–610. 2009. Springer-Verlag Berlin Heidelberg.
- M. Schmidt, J. Polowinski, J. Johannes, and M. A. Fernandez. An integrated facet-based library for arbitrary software components. In *ECMFA 2010*, LNCS 6138 proceedings, pages 261–276. 2010. Springer-Verlag Berlin Heidelberg.
- M. Voigt, A. Werstler, J. Polowinski, and K. Meißner. Weighted faceted browsing for characteristics-based visualization selection through end users. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12)*, pages 151–156. New York, NY, USA, 2012. ACM.
- U. Aßmann, A. Bartho, C. Bürger, S. Cech, B. Demuth, F. Heidenreich, J. Johannes, S. Karol, J. Polowinski, J. Reimann, J. Schroeter, M. Seifert, M. Thiele, C. Wende, and C. Wilke. DropsBox: the Dresden Open Software Toolbox. In *Software & Systems Modelling*. 2012.

Contents

Legend and Overview of Prefixes **xiii**

1 Introduction **1**

2 Background **11**

2.1 Visualisation 11

2.1.1 What is Visualisation? 11

2.1.2 What are the Benefits of Visualisation? 12

2.1.3 Visualisation Related Terms Used in this Thesis 12

2.1.4 Visualisation Models and Architectural Patterns 12

2.1.5 Visualisation Design Systems 14

2.1.6 What is the Difference between Visual Mapping and Styling? 14

2.1.7 Lessons Learned from Style Sheet Languages 15

2.2 Data 16

2.2.1 Data – Information – Knowledge 17

2.2.2 Structured Data 17

2.2.3 Ontologies in Computer Science 19

2.2.4 The Semantic Web and its Languages 19

2.2.5 Linked Data and Open Data 20

2.2.6 The Metamodelling Technological Space 21

2.2.7 SPIN 21

2.3 Guidance 22

2.3.1 Guidance in Visualisation 22

3 Problem Analysis **23**

3.1 Problems of Ontology Visualisation Approaches 24

3.2 Research Questions 25

3.3 Set up of the Case Studies 25

3.3.1 Case Studies in the Life Sciences Domain 26

3.3.2 Case Studies in the Publishing Domain 26

3.3.3 Case Studies in the Software Technology Domain 27

3.4 Analysis of the Case Studies' Ontologies 27

3.5 Manual Sketching of Graphics 29

3.6 Analysis of the Graphics for Typical Visualisation Cases 29

3.7 Requirements 33

3.7.1 Requirements for Visualisation and Interaction 34

3.7.2 Requirements for Data Awareness 34

3.7.3 Requirements for Reuse and Composition 34

3.7.4 Requirements for Variability 35

3.7.5 Requirements for Tooling Support and Guidance 35

3.7.6 Optional Features and Limitations 36

4 Analysis of the State of the Art 37
4.1 Related Visualisation Approaches 38
4.1.1 Short Overview of the Approaches 38
4.1.2 Detailed Comparison by Criteria 46
4.1.3 Conclusion – What Is Still Missing? 60
4.2 Visualisation Languages 62
4.2.1 Short Overview of the Compared Languages 62
4.2.2 Detailed Comparison by Language Criteria 66
4.2.3 Conclusion – What Is Still Missing? 71
4.3 RDF Presentation Languages 72
4.3.1 Short Overview of the Compared Languages 72
4.3.2 Detailed Comparison by Language Criteria 76
4.3.3 Additional Criteria for RDF Display Languages 87
4.3.4 Conclusion – What Is Still Missing? 89
4.4 Model-Driven Interfaces 90
4.4.1 Metamodel-Driven Interfaces 90
4.4.2 Ontology-Driven Interfaces 92
4.4.3 Combined Usage of the Metamodelling and Ontology Technological Space 94

5 A Visualisation Ontology – VISO 97
5.1 Methodology Used for Ontology Creation 100
5.2 Requirements for a Visualisation Ontology 100
5.3 Existing Approaches to Modelling in the Field of Visualisation 101
5.3.1 Terminologies and Taxonomies 101
5.3.2 Existing Visualisation Ontologies 102
5.3.3 Other Visualisation Models and Approaches to Formalisation 103
5.3.4 Summary 103
5.4 Technical Aspects of VISO 103
5.5 VISO/graphic Module – Graphic Vocabulary 104
5.5.1 Graphic Representations and Graphic Objects 105
5.5.2 Graphic Relations and Syntactic Structures 107
5.6 VISO/data Module – Characterising Data 110
5.6.1 Data Structure and Characteristics of Relations 110
5.6.2 The Scale of Measurement and Units 112
5.6.3 Properties for Characterising Data Variables in Statistical Data 113
5.7 VISO/facts Module – Facts for Vis. Constraints and Rules 115
5.7.1 Expressiveness of Graphic Relations 116
5.7.2 Effectiveness Ranking of Graphic Relations 118
5.7.3 Rules for Composing Graphics 119
5.7.4 Other Rules to Consider for Visual Mapping 124
5.7.5 Providing Named Value Collections 124
5.7.6 Existing Approaches to the Formalisation of Visualisation Knowledge . . 126
5.7.7 The VISO/facts/empiric Example Knowledge Base 126
5.8 Other VISO Modules 126
5.9 Conclusions and Future Work 127
5.10 Further Use Cases for VISO 127
5.11 VISO on the Web – Sharing the Vocabulary to Build a Community 128

6 A VISO-Based Abstract Visual Model – AVM 129
6.1 Graphical Notation Used in this Chapter 129
6.2 Elementary Graphic Objects and Graphic Attributes 131
6.3 N-Ary Relations 131
6.4 Binary Relations 131
6.5 Composition of Graphic Objects Using Roles 132

6.6 Composition of Graphic Relations Using Roles . . . 132
6.7 Composition of Visual Mappings Using the AVM . . . 135
6.8 Tracing . . . 135
6.9 Is it Worth Having an Abstract Visual Model? . . . 135
6.10 Discussion of Fresnel as a Related Language . . . 137
6.11 Related Work . . . 139
6.12 Limitations . . . 139
6.13 Conclusions . . . 140

7 A Language for RDFS/OWL Visualisation – RVL 141
7.1 Language Requirements . . . 142
7.2 Main RVL Constructs . . . 145
7.2.1 Mapping . . . 145
7.2.2 Property Mapping . . . 146
7.2.3 Identity Mapping . . . 146
7.2.4 Value Mapping . . . 147
7.2.5 Inheriting RVL Settings . . . 147
7.2.6 Resource Mapping . . . 148
7.2.7 Simplifications . . . 149
7.3 Calculating Value Mappings . . . 150
7.4 Defining Scale of Measurement . . . 153
7.4.1 Determining the Scale of Measurement . . . 154
7.5 Addressing Values in Value Mappings . . . 156
7.5.1 Determining the Set of Addressed Source Values . . . 156
7.5.2 Determining the Set of Addressed Target Values . . . 157
7.6 Overlapping Value Mappings . . . 158
7.7 Default Value Mapping . . . 158
7.8 Default Labelling . . . 159
7.9 Defining Interaction . . . 159
7.10 Mapping Composition and Submappings . . . 160
7.11 A Schema Language for RVL . . . 160
7.11.1 Concrete Examples of the RVL Schema . . . 163
7.12 Conclusions and Future Work . . . 166

8 The OGVIC Approach 169
8.1 Ontology-Driven, Guided Editing of Visual Mappings . . . 172
8.1.1 Classification of Constraints . . . 172
8.1.2 Levels of Guidance . . . 173
8.1.3 Implementing Constraint-Based Guidance . . . 173
8.2 Support of Explicit and Composable Visual Mappings . . . 177
8.2.1 Mapping Composition Cases . . . 178
8.2.2 Selecting a Context . . . 180
8.2.3 Using the Same Graphic Relation Multiple Times . . . 181
8.3 Prototype P1 *(TopBraid-Composer-based)* . . . 182
8.4 Prototype P2 *(OntoWiki-based)* . . . 184
8.5 Prototype P3 *(Java Implementation of RVL)* . . . 187
8.6 Lessons Learned from Prototypes & Future Work . . . 190
8.6.1 Checking RVL Constraints and Visualisation Rules . . . 190
8.6.2 A User Interface for Editing RVL Mappings . . . 190
8.6.3 Graph Transformations with SPIN and SPARQL 1.1 Update . . . 192
8.6.4 Selection and Filtering of Data . . . 193
8.6.5 Interactivity and Incremental Processing . . . 193
8.6.6 Rendering the Final Platform-Specific Code . . . 196

9 Application **197**
9.1 Coverage of Case Study Sketches and Necessary Features . . . 198
9.2 Coverage of Visualisation Cases . . . 201
9.3 Coverage of Requirements . . . 205
9.4 Full Example . . . 206

10 Conclusions **211**
10.1 Contributions . . . 211
10.2 Constructive Evaluation . . . 212
10.3 Research Questions . . . 213
10.4 Transfer to Other Models and Constraint Languages . . . 213
10.5 Limitations . . . 214
10.6 Future Work . . . 214

Appendices **217**

A Case Study Sketches **219**

B VISO – Comparison of Visualisation Literature **229**

C RVL **231**

D RVL Example Mappings and Application **233**
D.1 Listings of RVL Example Mappings as Required by Prototype P3 . . . 233
D.2 Features Required for Implementing all Sketches . . . 235
D.3 JSON Format for Processing the AVM with D3 – Hierarchical Variant . . . 238

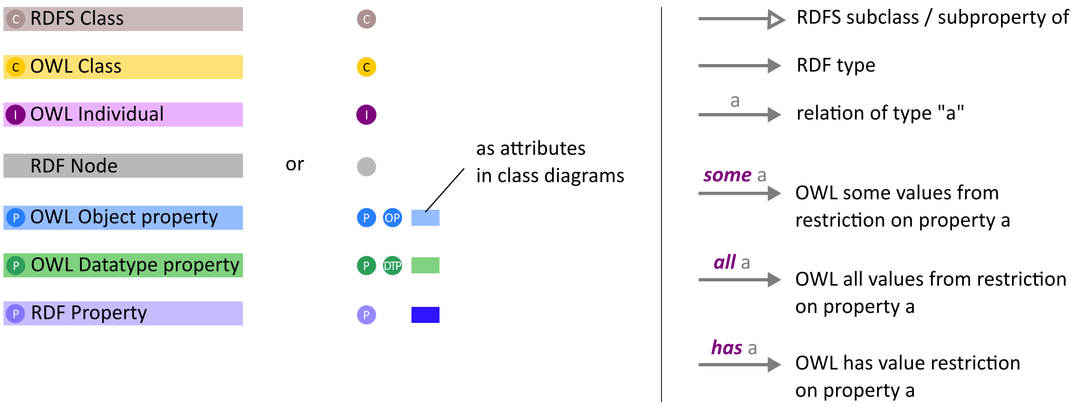
Bibliography **238**

List of Figures **251**

List of Tables **254**

List of Listings **257**

Legend



Overview of Prefixes and Namespaces

xsd:	XML Schema	http://www.w3.org/2001/XMLSchema#
rdf:	Resource Description Framework	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	RDF Schema	http://www.w3.org/2000/01/rdf-schema#
owl:	Web Ontology Language	http://www.w3.org/2002/07/owl#
dct:	Dublin Core Terms	http://purl.org/dc/terms/
dc:	Dublin Core Elements 1.1	http://purl.org/dc/elements/1.1/
spin:	SPARQL Inferencing Notation	http://spinrdf.org/spin#
sp:	SPIN SPARQL Syntax	http://spinrdf.org/sp#
smf:	SPIN Functions	http://spinrdf.org/spif/
spl:	SPIN Standard Library	http://spinrdf.org/spl#
fn:	XPath Functions	http://www.w3.org/2005/xpath-functions#
afn:	Jena ARQ Functions	http://jena.hpl.hp.com/ARQ/function#
rvl:	RDFS/OWL Visualisation Language	http://purl.org/rvl/
rvl-cs:	RVL SPIN-Constraints	http://purl.org/rvl/spin-constraints/
rvl-fb-cs:	RVL Fact-Based SPIN-Constraints	http://purl.org/rvl/fact-based-spin-constraints/
rexc:	RVL Example Commons	http://purl.org/rvl/example-commons/
viso:	Visualisation Ontology	http://purl.org/viso/graphic/
viso-graphic: / vg:	VISO/graphic module	http://purl.org/viso/graphic/
viso-data: / vd:	VISO/data module	http://purl.org/viso/data/
viso-facts:	VISO/facts module	http://purl.org/viso/facts/
common-shapes:	VISO Common Shapes	http://purl.org/viso/shape/commons/
bio-shapes:	Examples of shapes for the domain of Biology	http://purl.org/viso/addon/shapes/bio/
amino-acid:	Amino-Acids Ontology	http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl#
cito:	Citation Typing Ontology	http://purl.org/spar/cito/
obo:	Open Biomedical Ontologies (OBO)	http://purl.org/obo/owl/obo#
obo-rel:	OBO relations	http://purl.org/obo/owl/OBO_REL#
po:	OBO Plant Ontology	http://purl.org/obo/owl/PO#
zfo:	OBO Zebra Fish Anatomy Ontology	http://purl.org/obo/owl/ZFA
ro:	Requirements Ontology	http://purl.org/ro/ont#
ex:		No specific namespace but an example graph

Chapter 1

Introduction

The amount of information available on the World Wide Web and in isolated databases can hardly be managed, neither by machines, nor by humans. Data from social networks, sensor data and raw data from governments and industry, as currently postulated by the open data movement, will further increase the problem of information overflow.

One option to move from data masses to meaningful information is the formal description and linking of data sources using Semantic Web and Linked Data technologies. Ontologies written in Semantic Web languages, as recommended by the W3C, foster the sharing and linking of data by providing a means to formally define concepts and relations between these concepts. Due to the formal description and standardisation, it becomes easier for machines to connect data sources, infer implicit facts from existing knowledge and provide meaningful, semantically rich, data.

A second option is information visualisation, which combines the abilities of machines and humans to prepare, visualise, conclude and refine – in order to finally gain a graphic representation of the knowledge that was previously hidden in a data set. In contrast to the *textual* representation, the *visual* representation of facts allows humans to perceive information more directly, using the highly developed visual sense. At the core of the visualisation process is the *visual mapping*, i. e., the mapping of data variables or relations to visual means. Examples of such visual means are graphic attributes such as *colour hue*, *colour lightness* or *shape* and graphic relations between the objects in a graphic, e. g., *linking* or *containment*.

Relatively few efforts have been made so far on combining both of the options that we mention above – that is applying information visualisation to Semantic Web data, although the formality and rich semantics of the data make it an ideal candidate for visualisation. The general need for effective ontology visualisations has early been noted [SBLH06, KHL^{+}07] and also Paulheim and Probst conclude in their survey of ontology-enhanced UIs: »*There are clear research gaps in* [..] *advanced visualizations, and exploiting the possibilities of highly formal ontologies in user interfaces*« [PP10]. This is not to say that there are no approaches to visualising ontological data [GSGC08, CTW^{+}09, VPM13, LNHE14], and it also has to be stated that there are advanced visualisation design systems available, which support the user in visualising tabular, typically statistical data. An example for such a design system, is the commercial software Tableau [TAB]. However, we see many unsolved problems that can be summarised as follows:

First, specific, tailor-made visualisations of ontological data are still difficult and time consuming, since they have to be created manually. Many (semi-)automated solutions for generic visualisations exist, but they are usually limited to lists or node-link diagrams as illustrated in Fig. 1.1. In other cases, the available visual means (e. g., *colour hue* or *shape*) are not directly used to encode information, but only to structure and style a textual representation of the underlying data. Second, the additional semantics that the ontological data offers, are not exploited for guiding the end-user through visualisation tasks. And finally, once a good visualisation setting has been created, it cannot easily be combined with other existing settings or shared with

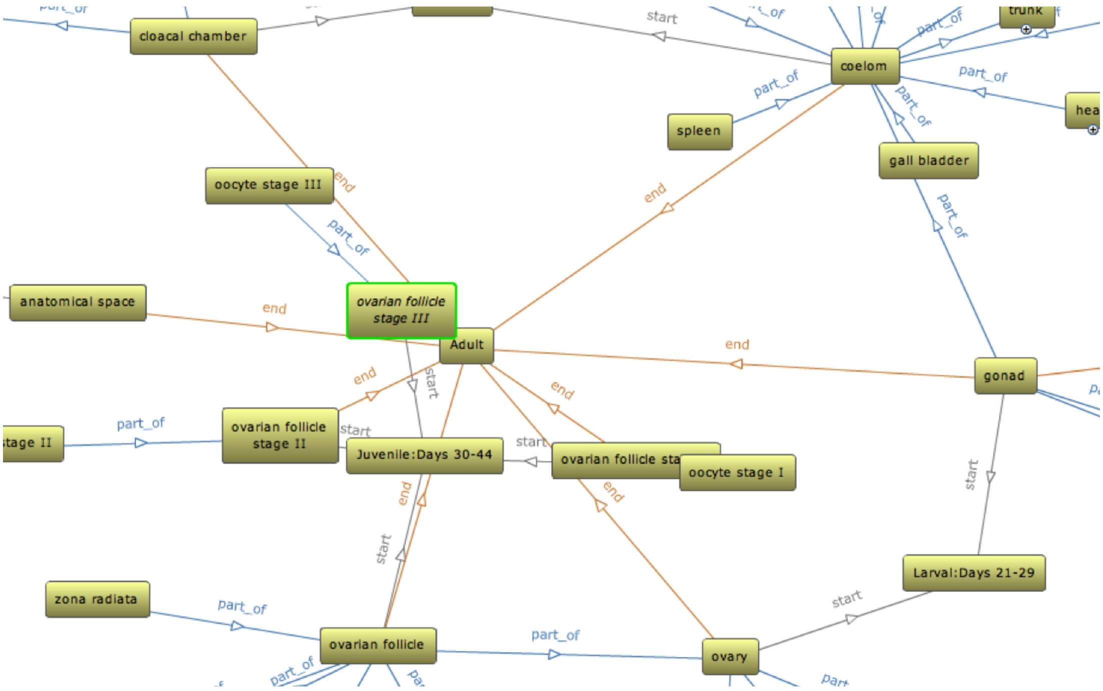


Figure 1.1: Example of a generic node-link representation of the *Zebrafish Anatomy Ontology* as created by the graph view on BioPortal[1]. To date, generic visualisation tools for ontological data often create node-link representations. In contrast, with the visualisation approach proposed in this thesis, we aim at tailor-made graphics as shown in Fig. 1.2.

other users. The lack of reuse also applies to visualisation knowledge (e. g., results from empiric visualisation studies). Such results could be used across different systems.

Trying to tackle these problems, the overall goal of this thesis is to present an approach that allows for a flexible, guided visualisation of ontological data, the creation of tailor-made, effective graphics and the reuse of visualisation settings. Instead of generating the above-mentioned generic graphics such as node-link diagrams or lists of resources, we want to enable domain-experts and other end-users to create fine-tailored graphics such as the one shown in Fig. 1.2. In contrast to many similar ontology representations known from ontology modelling environments – e. g., TopBraid Composer's[2] *Graph* view, the Protégé[3] plugins *OntoGraf*[4], *OWLViz*[5] or *Protégé VOWL* [LNB14] – we aim at using the whole palette of visual means. These include commonly used graphic attributes such as *colour hue* and (spatial) relations between graphic objects such as *linking* (e. g., used in node-link diagrams) and *containment* (e. g., used in Venn-diagrams) but also simple and effective, though more rarely used, ones, such as *proportional repetition* (e. g., used in »star rankings«), *proportional division* (pie charts), *clustering* and *line-up* of graphic objects. Furthermore, we do not aim at a visual notation for OWL like VOWL [LNHE14] or SOVA[6]. The fact that we aim at tailor-made graphics in a bottom-up approach [LNS06], distinguishes this work from top-down approaches such as

[1] BioPortal – http://bioportal.org (now http://bioportal.bioontology.org; accessed: 29.03.2010)
[2] TopBraid Composer – Ontology modelling environment. http://www.topquadrant.com, accessed: 12.04.2015.
[3] Protégé – Ontology modelling environment. http://protege.stanford.edu/, accessed: 16.04.2016.
[4] OntoGraf. http://protegewiki.stanford.edu/wiki/OntoGraf/, accessed: 16.04.2016.
[5] OWLViz. http://protegewiki.stanford.edu/wiki/OWLViz/, accessed: 16.04.2016.
[6] SOVA – Simple Ontology Vis. API. http://protegewiki.stanford.edu/wiki/SOVA/, accessed: 16.04.2016.

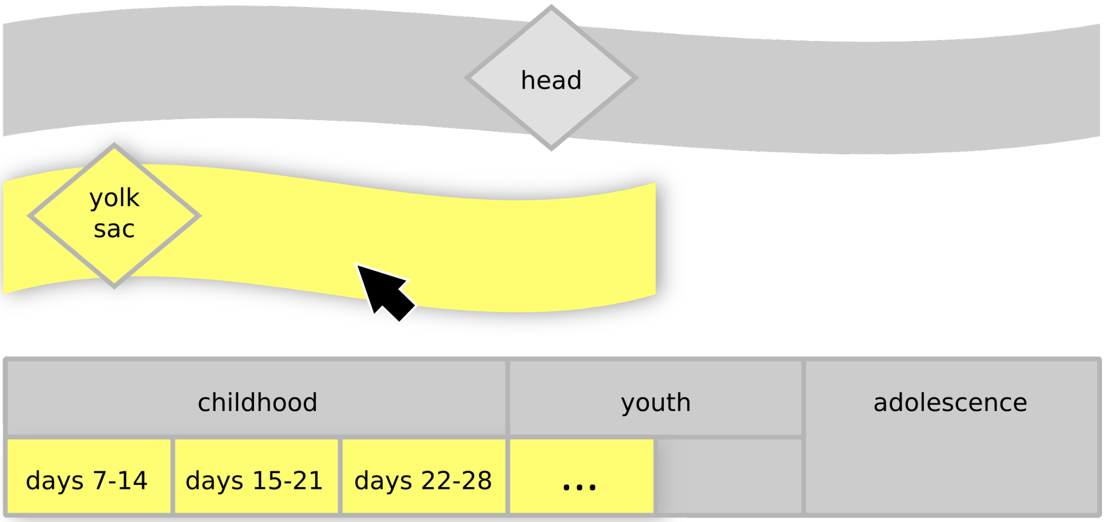


Figure 1.2: Example of a tailor-made graphic representing several concepts described in the *Zebrafish Ontology*. Various visual means such as *containment, line-up, width* and *position* are used to encode ontological relations such as *part-of* or the *start and end times* of phases during the fish's development. Multiple of these visual mappings are composed to yield a complex graphic. Pointing to parts of the zebra fish anatomy in the upper half of the graphic interactively highlights the corresponding development phases.

Vispedia [CTW$^+$09] or *VizBoard* [VPM13], which take the approach of building »mash-ups« by instantiating and connecting ready-made graphic types.

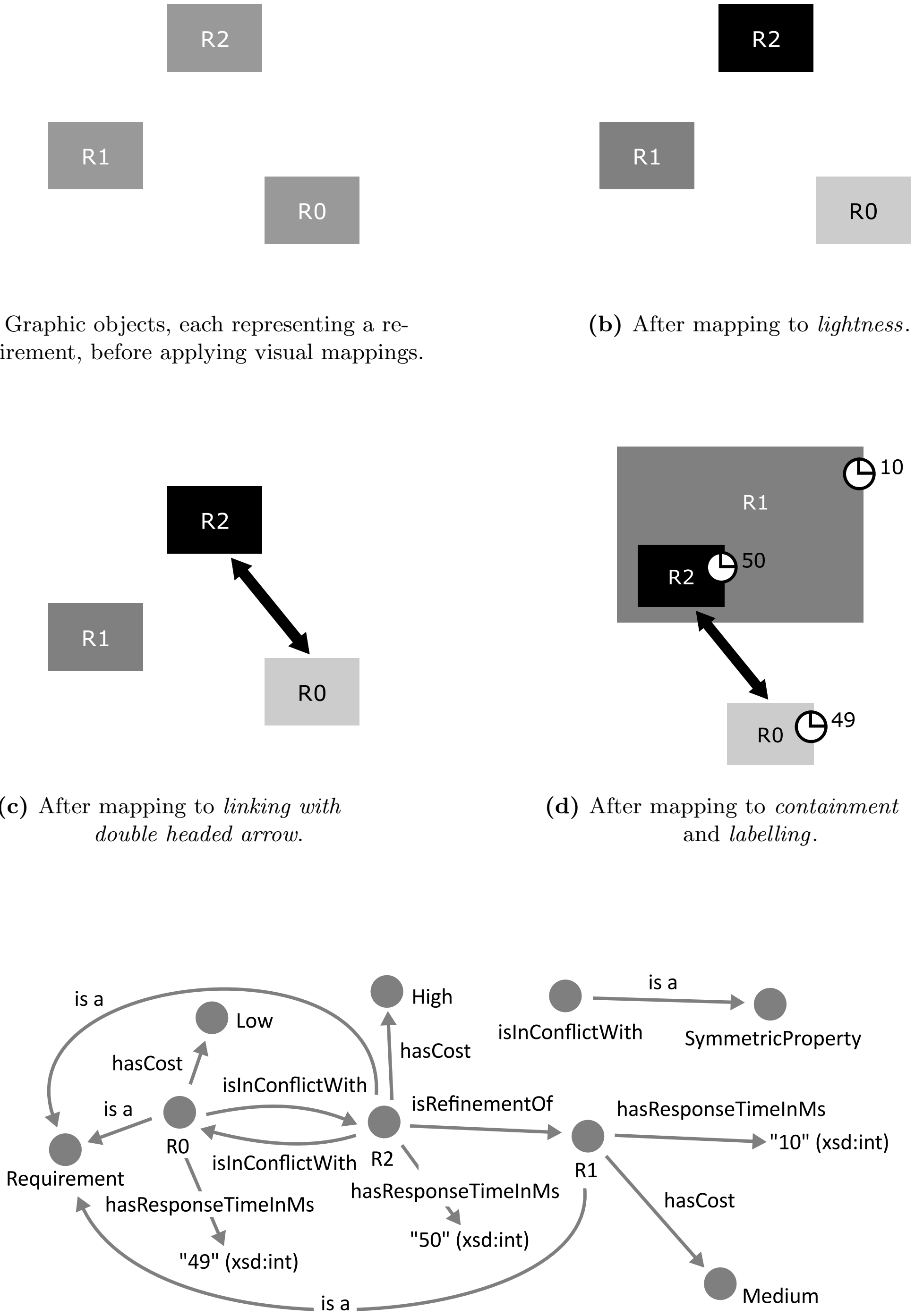
To receive an impression of how a user may interact with such a visualisation design system, and to further motivate our approach, we start with a concrete example taken from our case studies: Let us assume, we want to visualise data about software requirements that is described using the *Requirements Ontology* [STZ$^+$11]. In the following, we go through steps of an iterative, guided visualisation process as the system should offer it for visualising the requirements data. The resulting intermediate graphic after each step is shown in Fig. 1.3. During the interactive process, the design **system** and the **user** take action in turn:

- Initially, the **system** shows only a graphic object with its default appearance for each entity to be visualised (Shape = Rectangle, Colour = Grey, Position = Random . . . ; Fig. 1.3a).
- **System**: The following relations are relevant: »has priority«, »has response time in ms«, »is in conflict with«, . . .
- **User** selects »has priority«

Based on formally stored knowledge from the field of graphics and visualisation, the system can suggest effective visual means and guide the user:

- **System**: The following visual means are recommended: »lightness«, »position«, . . .
- **User** selects »lightness« . . .
- **System** shows Fig. 1.3b and suggests further relations . . .
- **User** selects »is in conflict with«

For the selected relation from the domain data – »is in conflict with« – information is available regarding the characteristics of this relation. For example, the system knows that it is a symmetric relation. Since symmetric relations are not directed, this should be reflected by the



(a) Graphic objects, each representing a requirement, before applying visual mappings.

(b) After mapping to *lightness*.

(c) After mapping to *linking with double headed arrow*.

(d) After mapping to *containment* and *labelling*.

(e) Generic representation of the same information encoded in (d).

Figure 1.3: Example for steps of an iterative, guided visualisation process using ontological data from the case study *Requirements Ontology* (a–d). The labels (*RO*, *R1*, *R2*) are shown from the beginning without requiring explicit labelling settings, because we assume that graphic objects are labelled by default with the local identifier (e. g., *RO*) of the represented resources (here requirements). For comparison, the last subfigure (e) shows a generic visualisation as typically provided by ontology editors.

graphic representation accordingly:

- **System** realises »is in conflict with« is a symmetric relation. Hence, it offers the *undirected* visual means »linking by line«, »linking by double-headed arrow«, ...
- **User** selects »linking by double-headed arrow«
- **System** updates the view and shows Fig. 1.3c

Following these steps, the system could offer adding further relations to the visualisation, such as »is refinement of«, »has response time in ms«, ... and the user could perform further mappings. After additional visual mapping steps the result could look like Fig. 1.3d, which is already a complex composed graphic employing five different visual means. Instead of a pure generic node-link diagram (cf. Fig. 1.3e), we replaced some links and nodes by alternative ways of graphic representation, thereby obtaining a less crowded diagram.

This example already demands some of the requirements[7] that we have for our visualisation approach: First, we need a guidance system for end-users that is aware of types and characteristics of the relations used in the source data and that has access to a collection of formalised, machine-readable visual means such as *colour* and *containment*. Second, this graphic vocabulary needs to be formally defined. Third, for recommending visual means, the system also requires access to a collection of perceptual facts on visualisation.

A second set of requirements emerges, when we have a closer look at what different actors may want to do with the visual mappings, once they have defined a set of mappings that suits their purposes. Let us assume that the user wants to share the visualisation settings she created with a colleague. The colleague has similar needs of visualising requirements (which have also been specified with the Requirements Ontology), but works with different visualisation software on a different platform. Additionally, she uses an extension of the requirements ontology that introduces a set of missing relations she requires. From this use case, we derive further requirements: To support reuse and sharing of the visual mappings that we define with our system, these mappings should be stated explicitly, they should be composable and usable on different platforms. In order to further support the sharing of visual mappings, existing standards such as the W3C languages for the Semantic Web – RDF(S), OWL and SPARQL – should be used wherever possible.

Having presented current problems in visualising ontological data and having motivated our general goal, in the following, we briefly outline how this thesis contributes to a solution:

Contributions

In this thesis, we make the following main contributions, each of which corresponds to a chapter in the remainder:

C-1 Our first contribution is the OGVIC approach to **Ontology-Driven, Guided Visualisation Supporting Explicit and Composable Mappings** (Chapter 8), which we conceptually describe in Fig. 1.4. Using this figure, we briefly introduce the overall approach and point to additional contributions that were created as necessary prerequisites.

We assume that ontological data described in RDFS/OWL is selected from a Linked Data source or read from local files (a, b). Once available to the visualisation system, this data can be filtered at any time to change the subset of filtered data that will be visualised (c, d). Selection and interactive filtering are not in focus of this thesis, since a variety of mechanisms have already been proposed for these process steps such as faceted browsing and other visual query mechanisms [ODD06, VWPM12, HZL08].

[7] We give a complete list of additional visualisation cases, concrete problems, research questions and requirements, as well as a detailed description of actors and use cases in Chapter 3 (Problem analysis).

Following the principle of the Model Driven Architecture[8] (MDA), we suggest to generate an *Abstract Visual Model* – the AVM – in a next step (e, f). In terms of the MDA, the AVM is a *Platform-Independent Model.* It consists of graphic objects and relations that have been formalised in the *Visualisation Ontology* – the VISO (g1–g3). The information on which transformations (including the visual mapping) from the domain data (c) to the AVM (f) are to be performed is described in a fourth model – the *RVL mappings* model (h), which is based on the declarative RVL language (introduced further below). Finally, in a rendering[9] step (j), the actual concrete *rendered graphic* (k) for a final platform such as SVG, HTML or X3D is created from the AVM. One reason for this extra abstract graphic model becomes apparent, when we look at the remaining process step – the *guided editing of visual mappings* (i), which is at the core of this thesis: During this step – while guiding the user through the process of creating and editing a set of visual mappings – not only the available data and the possible visual means need to be taken into consideration; also the AVM needs to be available for introspection. This is because additional mappings may be constrained by existing ones. This would be impossible if we had only access to the resulting final (SVG) graphic.

As foundations for our approach, multiple technologies, ontologies and languages have been developed and published as additional contributions of this thesis.

C-2 The developed ontologies comprise the **VISO ontology** [PV13, VP11] (g1–g3) that was already mentioned above, consisting of multiple modules such as *VISO/graphic* (formalising graphic concepts), *VISO/data* (for formalising data-characteristics) and *VISO/facts* (offering vocabulary to describe knowledge gained from visualisation research). Additionally, a default fact base *VISO/facts/empiric* (g4) was created to store facts on the effectiveness of visual means as described in visualisation literature (Chapter 5).

C-3 The **principle of the Abstract Visual Model (AVM)** can be seen as a contribution on its own, since it is the first approach to formalise the various, complex relations between graphic objects following the observations from Engelhardt [vE02]. The AVM allows for modelling roles of graphic objects and, thereby, lays the foundation for a precise composition of multiple visual mappings (Chapter 6).

C-4 A further contribution and foundation of our approach is the **RDFS/OWL Visualisation Language (RVL)** [Pol13]. RVL allows for composable, declarative mappings that can be shared among users and systems and may be stored along with the RDF domain data. An RVL mapping could, for example, explicitly define a visual mapping of an RDF property to a specific visual means, such as: »*Map the property dc:hasPart to the graphic relation viso-graphic:Containment_Relation*« (Chapter 7).

Fig. 1.4 also clarifies why we call our approach *ontology-driven* and shows how ontologies are used to drive the visualisation process at various places: First, as a rich source of information on the data characteristics (c), second, as a means to formally describe the vocabulary for building abstract graphics (g1), and third, as a formal knowledge base to store facts on the effective use of visualisations (g4). We give a precise definition of »ontology-driven« in Sect. 4.1.2.

C-5 Besides the contributions shown in Fig. 1.4, we made a **detailed analysis of the state of the art** in the field of visualisation approaches and languages used for visualisation and RDF-presentation, which we see as an additional contribution of this thesis (Chapter 4). For the analysis of related visualisation approaches, a set of 29 criteria was established and applied to compare ten approaches in detail. Existing work only covers a subset of

[8] Model Driven Architecture. http://www.omg.org/mda/, accessed: 12.12.2015.

[9] By *rendering*, in this thesis, we do not refer to the process of creating a rasterised image, but we use the term in a broader sense for a transformation to a concrete (graphic) format.

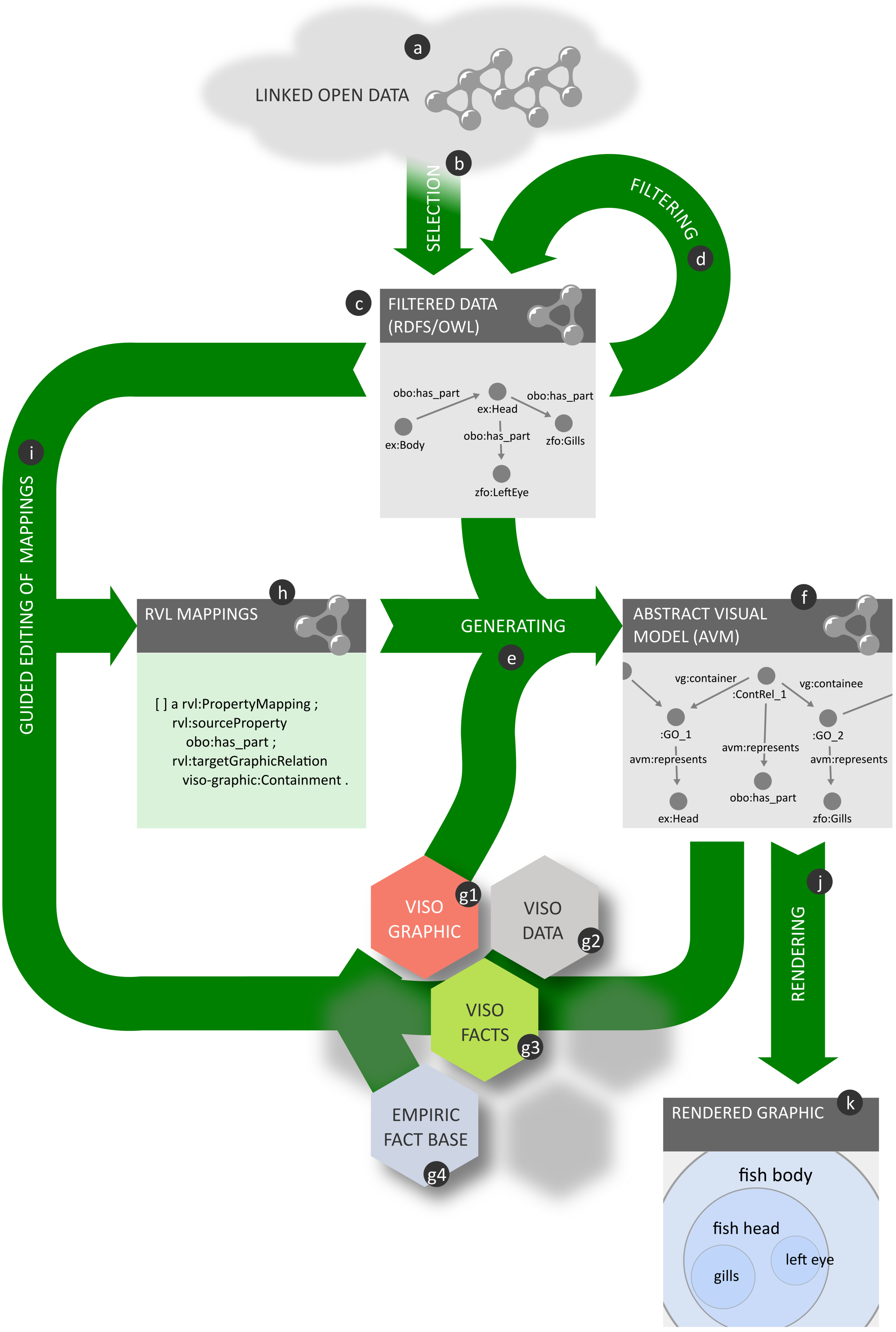


Figure 1.4: The principle of OGVIC – Schematic overview. This figure illustrates the complete OGVIC approach showing the main process steps and models (a–k), which are referenced in this section. For each of the three models – the »Filtered (domain) data« model, the »RVL mappings« model and the »Abstract Visual Model« – a brief example is given using ontological data from the *Zebrafish Ontology*.

the four aspects *Visual mapping*, *Composability*, *Guidance* and *Ontology-driven*. Fig. 1.5 summarises the results from our analysis of related visualisation approaches and shows the combination of aspects that makes the OGVIC approach unique: The approach is the first to combine ontology-driven guidance with the possibility to define composable visual mappings.

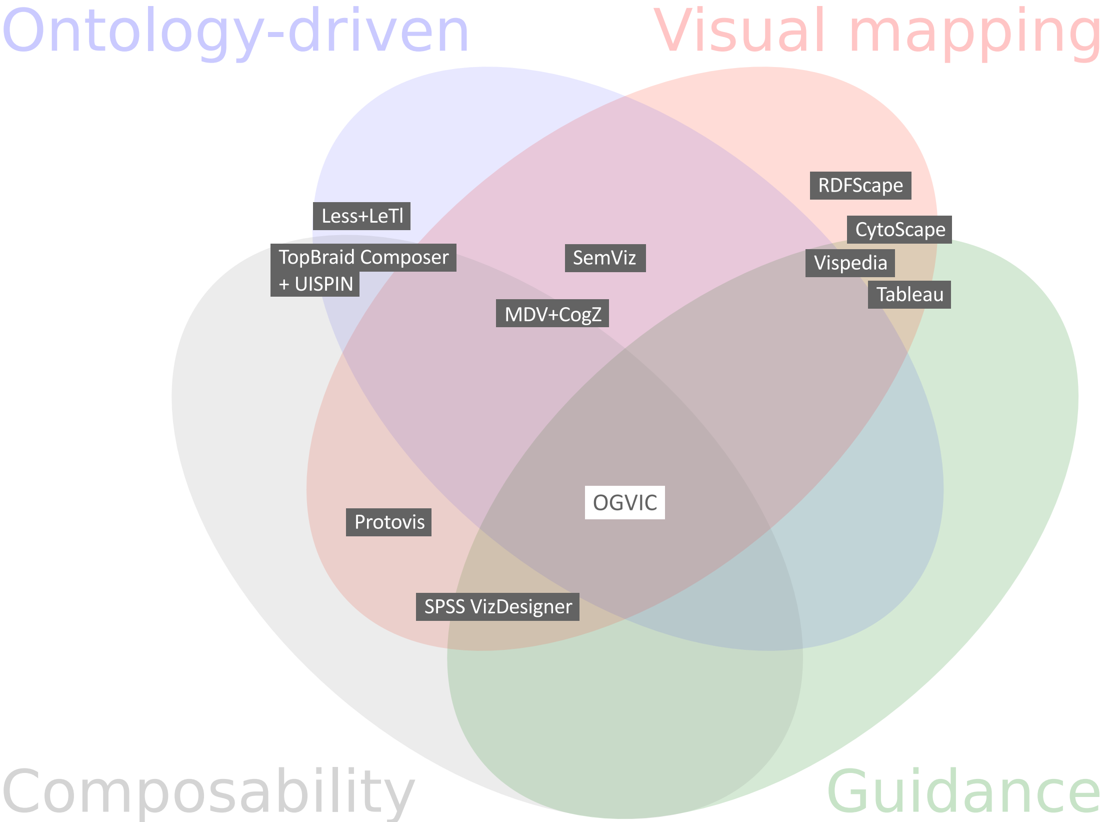


Figure 1.5: Unique selling points of OGVIC – The approach is the first to combine ontology-driven guidance with the possibility to define composable visual mappings. Existing work only covers a subset of the four aspects *Visual mapping*, *Composability*, *Guidance* and *Ontology-driven*.

C-6 Finally, we compare three prototypical implementations that have been created[10] to cover the essential parts of the OGVIC approach and show its feasibility. Two of them are embedded into existing editors for RDF data – TopBraid Composer and OntoWiki. A third one has been built from scratch. While no prototype covers all aspects of the OGVIC approach, our first two prototypes serve demonstrating the constrained-based guidance for defining RVL mappings as well as the rule-based interpretation of RVL mappings. The last one puts the focus on interpreting as much of the RVL specification as possible and rendering concrete interactive (web) graphics (Sect. 8.3–8.5).

[10] The OntoWiki-based prototype has been developed by Pooran Patel in student work supervised by the author.

Is this approach limited to Ontologies or RDFS/OWL?

The OGVIC approach is exemplified in the RDF technical space, which offers the benefit that the graphic knowledge we formalised as an OWL ontology can directly be accessed and the contained implicit knowledge can be exploited. As a second benefit, we can directly use terms of the graphics vocabulary for building the Abstract Visual Model (AVM) without translating back and forth between technological spaces. In order to concentrate on the relevant research questions, we completely stay within the (RDF-based) ontology technological space in the following chapters. However, the general ideas described in this thesis as the OGVIC approach are not bound to RDF-based ontological data and could at least partly be transferred to other models than ontologies (e. g., *Ecore* models; Sect. 2.2.6), once stable bridging technology is available (cf. the work of Gašević et al. [GDD05] and Aßmann et al. [AEWW13a] for first approaches in this direction).

Outline of the Thesis

This thesis is structured as follows: After this introduction, where we have motivated our approach by a concrete example and pointed to our main research goals, questions and contributions, we provide background information on enabling technologies used in this thesis for readers who are not familiar to the field of Semantic Web or Information visualisation (Chapter 2).

We then describe each of the contributions of this thesis in its own chapter: First, we present a detailed problem analysis (Chapter 3) and an analysis of the state of the art (Chapter 4). The next chapters present our formalisation of terms in the field of graphics, the Visualisation Ontology (VISO), in Chapter 5, the Abstract Visual Model (AVM) in Chapter 6 and the RDFS/OWL Visualisation Language (RVL) in Chapter 7.

After having presented all necessary foundations for our approach, we introduce the OGVIC approach to Ontology-Driven, Guided Visualisation (Chapter 8) and briefly present prototypical implementations as well as lessons learned from building these prototypes. Chapter 9 presents the results of applying the prototypes to ontologies from our case studies. Finally, in Chapter 10, we conclude from our findings and discuss current limitations of OGVIC as well as future work.

Chapter 2

Background

In the following, we briefly introduce the fields of visualisation, ontologies and guidance and point to how the OGVIC approach relates to these fields. Since this chapter mainly serves as a place to look up basic knowledge for readers lacking some of this background, readers may skip sections accordingly.

2.1 Visualisation

In the first section of this chapter, we define basic terms from the field of visualisation such as *visualisation* itself and *visualisation design system*. Further, we give reasons for using visualisation and introduce existing visualisation models and architectures. Finally, we distinguish visualisation languages from well-established style sheet languages and summarise lessons learned from the design of these languages.

2.1.1 What is Visualisation?

The term *visualisation* often refers to both – the process and the product. On the one hand visualisation can be seen as the mapping from data to visual form. This mapping includes transformation and often human interaction, so it is not a static document, but a process. On the other hand, often also the product of a visualisation process is called a visualisation. As the product often allows further interactive settings, especially in information visualisation, the process of visualisation continues with the product. This way the product and the process are closely intertwined. For clarity, we refer to the product of a visualisation process as a *graphic representation* or short *graphic* in this thesis. Maps are one of the oldest examples of graphic representations. Charts, tree maps, class diagrams, timelines and interactive visual browsers are other examples.

Definition 1 (Visualisation)
The process of creating visual graphic representations from data.

Additionally, *visualisation* stands for a field of research. Since we are formulating an approach to visualise abstract information stored in ontologies – rather than raw physical data – our work can be assigned to the field of *information visualisation* (InfoVis). If we consider that ontologies store knowledge (and not only information), we could even more precisely classify our approach under the less popular term of *knowledge visualisation.* A distinction of data, information and knowledge is given in Sect. 2.2.1.

2.1.2 What are the Benefits of Visualisation?

The question »Why should we use visualisation?« is easier to answer than the question what visualisation actually is: It amplifies cognition. This is widely accepted, for example, according to Card et al. [CMS99], visualisation amplifies cognition by

(a) increasing the memory and processing resources available to the users,

(b) reducing search for information,

(c) using visual representations to enhance the detection of patterns,

(d) enabling perceptual inference operations,

(e) using perceptual attention mechanisms for monitoring,

(f) encoding information in a manipulable medium.

The authors list references and give examples, which support each of the statements. For example, the increase of memory and processing resources (a) can be explained by the fact that some graphic attributes can be processed faster and in parallel, compared to text, which needs to be processed serially. The reduced search (b) is explained by the fact that visualisations can often »*represent a large amount of data in a small space*«, a phenomenon that has been described by [Tuf83] as the *Data–Ink Ratio*. The reader is referred to Card et al.'s work for further reading.

2.1.3 Visualisation Related Terms Used in this Thesis

While we give detailed definitions of visualisation related terms in Chapter 5, we would like to give the reader a brief introduction to two terms that will frequently be used in the following chapters already: *visual means* and *visual structure*. In the less technical chapters of this thesis, we use the term *visual means* for attributes (e. g., *colour*, *width*, *x-position*) as well as relations between graphic objects such as *linking*, *overlapping*, *containment*. The latter group, the relations, form larger constructs that are often referred to as *visual structures*. The *main visual structure*, i. e., the one that dominates a graphic, is sometimes also called *visual paradigm*.

2.1.4 Visualisation Models and Architectural Patterns

Several abstract models with different focuses were developed to allow for a better understanding of the visualisation process. The pipeline models of Mackinlay [Mac86a], and Haber and McNabb [HM90] describe visualisation as a process consisting of a series of transformation steps that convert data into a displayable image. Other researchers adopted this model and extended it, e. g., with human interaction and tasks [CMS99], focused on modelling data states [Chi00] or the coordination of different views [BRR03].

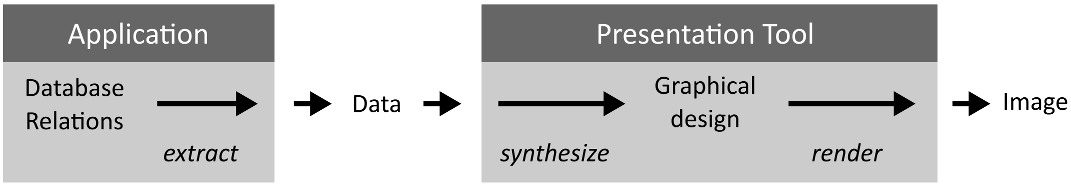


Figure 2.1: A pipeline model as used by Mackinlay to demonstrate a linear process of generating graphics, redrawn after Mackinlay [Mac86a].

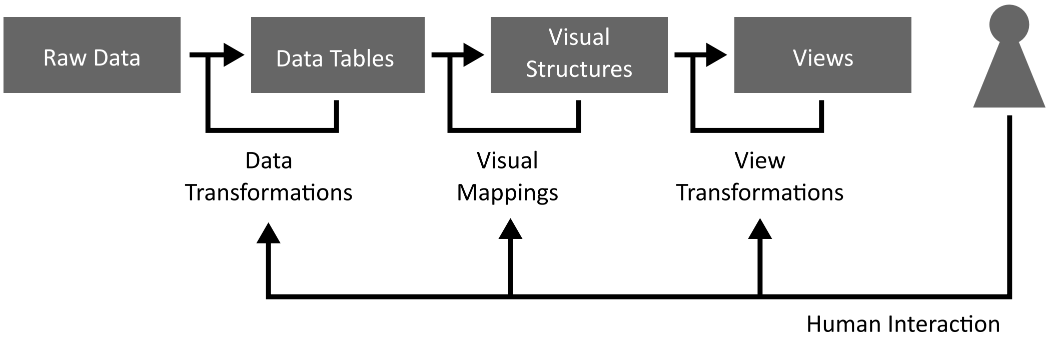


Figure 2.2: Visualisation Reference Model, redrawn after Card et al. [CMS99].

Pipeline Models for Visualisation

Fig. 2.1 shows a typical linear pipeline model, as used by Mackinlay [Mac86a] to illustrate the stepwise visualisation process taken by the tool he developed. Extracted data is used to synthesise an intermediate *Graphical design*, which is then rendered to the final *Image*. Mackinlay already noted that for »difficult design problems« »feedback loops« are required, i. e., the model needs to be extended by human interaction. Also Haber and McNabb [HM90] describe three transformation steps, which »convert« raw data to a *Displayable Image* (not shown). However, they emphasise the fact that data variables are mapped to graphic attributes by calling the central transformation step *Visualisation Mapping*.

The Visualisation Reference Model – Modelling Interaction

Interaction is a defining characteristic of information visualisation:

> » Information visualisation is about the not just creation of visual images, but also the interaction with those images in the service of some problem.
>
> *Stuart Card [CMS09]* «

As examples of interactive visualisation structures Card et al. [CMS99] list *Dynamic queries*, *Magic lens*, *Overview and detail*, *Linking and brushing*, *Extraction and comparison* and the *Attribute explorer*.

As an extension of the pipeline models with respect to human interaction, Card et al. developed the *Visualisation Reference Model* (Fig. 2.2). The user can interact with the system by modifying all three transformation steps – data transformations, visual mappings and view transformations. Navigation in the scene and changing the perspective are examples of view transformations, which can be triggered from within an existing view, but also the data transformations and the visual mappings can be modified interactively.

While the model of Card et al. became popular as the *Visualisation Reference Model* in the InfoVis community, visualisation reference models in general have been discussed already in 1993 by Butler et al. [BAB+93]. A further reference model with a different focus is the *Data State Reference Model* developed by Chi [Chi00], which emphasises the intermediate results in the visualisation process and allows for modelling multiple operations on each of four data stages (not shown). The intermediate stage between data and views is not called *Visual Structures*, but *Visualization Abstraction* by Chi – similarly, we call the intermediate graphic model that we introduce in Chapter 6 the *Abstract Visual Model* (AVM).

The Reference Model Pattern for Visualisation

Analysing the models of Chi and Card et al., Heer and Agrawala [HA06] described the *Reference Model* pattern (Fig. 2.3) as a visualisation specific software design pattern [GHJV94], which captures the essential structure of a software architecture that supports interactive visualisation. The authors note that the Reference Model pattern can be interpreted as a »*tiered version of MVC* [Model View Controller, [Bur92]]*, with the model divided into separate abstractions for the data and visual properties*«.

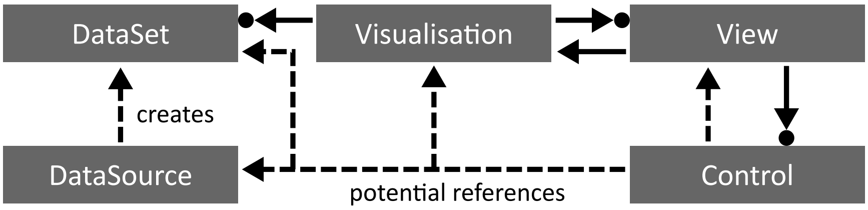


Figure 2.3: The Reference Model pattern for visualisation, redrawn after Heer and Agrawala [HA06].

2.1.5 Visualisation Design Systems

The OGVIC approach presented in this thesis is an approach to create visualisation design systems for ontological data. Therefore, we define the term *Visualisation Design System* and summarise classifications of these systems.

Definition 2 (Visualisation Design System) *A system that takes data as input and either creates a visualisation automatically or helps the user to do so.*

Lange, Nocke and Schumann [LNS06] distinguish systems following a *top-down* approach from those following *bottom-up* approaches. Top-down approaches use templates of complex, »predesigned« [Mac86a] graphic representations, which are then populated with data. While this allows for offering proven and tested graphics for frequently occurring scenarios, the extensibility of these approaches is limited. Bottom-up approaches synthesise graphics, for example using an algebra, which allows for creating graphics in a more flexible way. However, composing multiple visual means is challenging, since many rules have to be considered in order to achieve useful graphics. The OGVIC approach describes a bottom-up approach for creating graphics.

Visualisation design systems can further vary with respect to their degree of automation and the supported customisation times [Bul08]. We discuss visualisation systems and approaches in Sect. 4.1.1 in detail.

2.1.6 What is the Difference between Visual Mapping and Styling?

Visual mapping differs from styling. For styling, a wide-spread declarative language for the definition of styles exists – the Cascading Style Sheets (CSS) – which are used in combination with HTML and other XML languages such as SVG. We define *Styling* for the scope of this thesis as follows:

Definition 3 (Styling) *The process of adding styles to a structured document, where styles are variations of how a structured document and its parts are presented.*

Styles usually concern visual properties, for example, variations of font-face, font-colour, border-width and position, but they may also be created for aural properties such as the intonation or volume. The W3C states on its website [W3C13] that style sheets, which bundle a set of styles into a file, »*describe how documents are presented on screens, in print, or perhaps how they are pronounced.*« Typical situations for defining styles include the following and are always anchored to parts of the document structure (in this case an HTML structure):

- Set the font style of a specific DIV-element to »italic«.
- Set the border-width of all DIV elements to »1.5 cm«.
- Set the style of anchors to »underline«.

Visual mappings are at the core of the visualisation process. We define them as follows:

Definition 4 (Visual Mapping) *Also referred to as »visual encoding«. During the visual mapping process, data relations and values are mapped to (encoded as) visual relations and values. Besides for the mapping process, we also use the term visual mapping for descriptions of visual mappings that state source and target in a declarative way.*

We can distinguish three types of visual mappings:

a) Manual mapping to constants Here a data relation (e. g., *age*) or a value (e. g., *40 years*) is mapped to a constant visual attribute value (e. g., *red*).

b) Manual mapping to visual relations Here a data relation (e. g., *part of*) is mapped to a constant visual relation (e. g., *containment* or *linking*).

c) Dynamic, value-depending mapping o visual attributes A binding between values of data relations and graphic attributes *dynamically depending* on the relation between a set of values (e. g., *map the range of age values [0,100] to a range of colour values between red and yellow*).

The assignment of styles with CSS is different from the definition of visual mappings for RDF properties for two reasons: First, styles are applied to parts of a structured document, while visual mappings directly refer to data relations and data values. Second, styles are usually not dynamic and value-dependent. Similar to mapping type (a), they define constant values – but usually with the intention to achieve a more aesthetic or readable presentation, rather than to encode data values. A calculation of values, type (c), could only be done with XSL, not with CSS (both languages will be compared in detail in the next section). The problem that we cannot attach styles directly to RDF properties is targeted by Fresnel, which we introduce in Sect. 4.3.1. Fresnel allows for describing how an RDF graph should be turned into a document structure and, in a second step, then allows for assigning styles to the elements of the newly created structure. To define visual mappings, we do not need to first transform a knowledge graph into a document shape.

2.1.7 Lessons Learned from Style Sheet Languages – Separation of Concerns

Having pointed out the difference between styling and visual mapping, it is still worth looking at style sheet languages in more detail to learn from these languages. Therefore, we briefly introduce and compare the style sheet languages CSS and XSL. By separating the styling information from the rest of the document (structure and content), device independence is supported. Using a selection mechanism, the document parts that are to be styled can be selected and then be used in style rules.

Cascading Style Sheets (CSS), initially CHSS, the H standing for HTML, were invented to prevent HTML from turning into a full »page description language« [Lie05]. They are now a W3C standard for defining presentation, not only of HTML webpages, but also of XML documents, vector graphics (SVG) and UIs (XUL[1]). The term *cascading* refers to a priority scheme that determines, which style to select in case of multiple applicable style rules. For example, styles are inherited (propagated) along the box structure of the document, i. e., inner elements will inherit style values from outer elements unless an explicitly defined style (with higher priority) overrides the more general, inherited one. Cascading also allows for combining styles from multiple style sheets, which distinguishes it from other style languages and lends itself for the use on the web.

	CSS	XSL
Declarative? [Lie05]	yes	yes
Can be used with HTML?	yes	no
Can be used with XML?	yes	yes
Transformation language?	no	yes
Loss of machine-readable semantics? [Lie05]	no	yes
Syntax	CSS	XML

Table 2.1: Comparison of XSL and CSS, extended after a table from the W3C's »Web Style Sheets home page« [W3C13].

Extensible Stylesheet Language (XSL) is actually a bundle of W3C-recommended languages, consisting of XSL-Formatting objects (XSL-FO) – a page description language, Extensible Stylesheet Language Transformations (XSLT) as a language for transformations between XML documents and X-Path as a query language for (XML) trees, which provides the selection mechanism in XSL.

While both CSS and XSL are declarative and use the same formatting principles, important differences between the two languages exist: A major difference is that XSL follows a transformational approach, whereas CSS is interpreted by the presenting client. Lie points to the problems that occur with transformation-based style sheet languages: Applied on the server side »*there will be a loss of semantics since the transmitted content is at a lower level of abstraction*«. Applied at the client side he argues that »the browser will not be able to support progressive rendering of content where content is displayed in small chunks as the document is downloaded. Since the transformation may specify that the last element in the logical structure should come first in the presentational structure, the whole document must be downloaded before the transformation can take place«. Another difference between CSS and XSL is that XSL uses the syntax of XML, while CSS defines its own syntax. Table 2.1 summarises differences and commonalities between CSS and XSL.

2.2 Data

In this section, we define basic terms from the field of data such as what is *structured data* and how it relates to *ontologies* and *ontological data*. Further, we give a brief overview of the *Semantic Web* technologies mentioned in this thesis and clarify what is meant by *Linked Open Data*.

[1] XML User Interface Language. https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/, accessed: 03.11.2015.

2.2.1 Data – Information – Knowledge

Since we speak of visualising structured *data* while many of the approaches we review in this thesis belong to the field of *information* visualisation and some even use the term *knowledge* visualisation, we briefly need to clarify these three terms, which are often used interchangeably in everyday language, but also in scientific literature. Chen et al. [CEH$^+$09] tried to resolve the terminological confusion by introducing a set notation, which distinguishes between data, information and knowledge in the *perceptual and cognitive space* (P) on the one hand and the *computational space* (C) on the other hand. For the perceptual space Chen et al. reuse the definitions from Ackoff [Ack89]:

P_{data}	$P_{information}$	$P_{knowledge}$
symbols	data that are processed to be useful, providing answers to ‘who’, ‘what’, ‘where’, and ‘when’ questions	application of data and information, providing answers to ‘how’ questions
C_{data}	$C_{information}$	$C_{knowledge}$
computerized representations of models and attributes of real or simulated entities	data that represents the results of a computational process, such as statistical analysis, for assigning meanings to the data, or the transcripts of some meanings assigned by human beings	data that represents the results of a computer-simulated cognitive process, such as perception, learning, association, and reasoning, or the transcripts of some knowledge acquired by human beings

2.2.2 Structured Data

Structured data is the prerequisite for visualising data, since we need to identify some structure in the data that we can map to visual means or use in interactions such as filtering and faceted browsing [Kar13]. This structure is defined by means of a data model such as a database schema, an XML schema – or an ontology. A definition of *ontology* will follow in the next subsection.

Often three classes of data are distinguished – *unstructured data*, *semi-structured data* and *structured data*. While some agreement exists on the distinction between unstructured data (e. g., text documents, images) and structured data, the definition of semi-structured data is harder than it might appear at first sight. Abiteboul lines up the following main characteristics of semi-structured data (among others): »*the structure is irregular*«, »*partial*«, and »*implicit*«, »*the schema is very large*« and »*the distinction between schema and data is blurred*« [Abi97]. However, according to Buneman [Bun97], it is not always differentiated between *semi-structured data* and *unstructured data* in literature.

A related question concerns the formality of the data: Instead of a three-fold classification, Uschold et al. [UG04] suggest a continuum of languages for data modelling (Fig. 2.4), ranging from glossaries and thesauri via XML and database schemas to frames, description logics and first-order logic. From left to right they describe an increasing »*amount of meaning specified*« and an increasing degree of formality, which leads to a reduction of ambiguity. Further, they also describe an »*increasing support for automatic reasoning*« (i. e., the ability to infer new explicit facts from implicit knowledge) towards the right end of the continuum. Formal ontologies, stated in logic languages, are at the rightmost end of the spectrum.

Rauschmeyer [Rau10] characterises structured data by five criteria: First, the *flexibility of the schema*, with respect to annotation and schema evolution. Second, *globally unique symbols*, to ensure all data on a given concepts can be gathered from distributed sources and third, simple *composability* of these distributed data. Fourth, *links between entities* to organise entities by cross-referencing and fifth, *standardised exchange formats* to allow for applying external tools. Ontologies written in standard ontology languages, as described in the next subsections, fulfil these criteria.

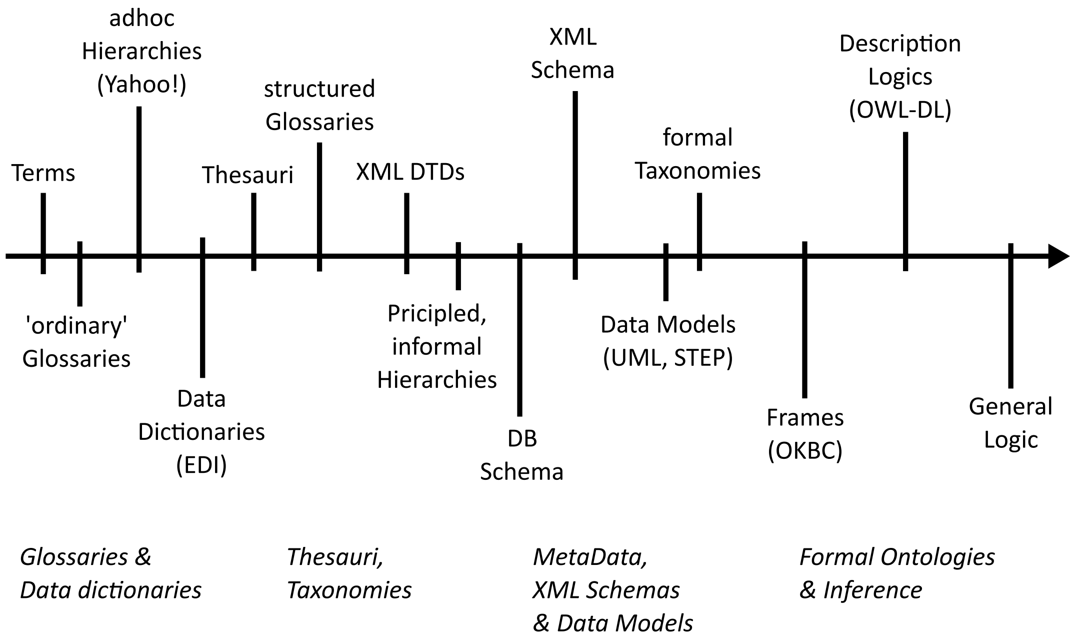


Figure 2.4: Continuum of data modelling languages (simplified, after Uschold et al. [UG04]). Uschold et al. refer to all these languages as ontologies, while we only use the term *Ontology* where we refer to *Formal Ontologies*, which are positioned at the rightmost end of the spectrum.

2.2.3 Ontologies in Computer Science

An often cited definition for ontologies in computer science was given by Gruber [Gru93]: »*An ontology is an explicit specification of a conceptualization.*« Guarino et al. [GOS09] give a good overview of how this broad definition was combined by Studer et al. [SBF98] with the definition from Borst [Bor97] to include the two additional aspects of being *formal*, i. e., machine-readable, and *shared*. For the readers of this thesis, we also consider the following excerpt of a more verbose definition of *Ontology* helpful, which was given by Gruber in 2009 [Gru09] pointing out the differences and commonalities between an ontology and a database:

> » […] an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. In the context of database systems, ontology can be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modelling knowledge about individuals, their attributes, and their relationships to other individuals. Ontologies are typically specified in languages that allow abstraction away from data structures and implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the »semantic« level, whereas database schema are models of data at the »logical« or »physical« level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services. […]
>
> *Thomas R. Gruber [Gru09]* «

Ontologies usually consist of two components, the **T-Box** (terminological component) containing classes, relations and their properties and the **A-Box** (assertion component) containing statements about instances or individuals. While the OGVIC approach also supports the visualisation of the T-Box, we sometimes use the term *ontological data*, or *ontology instance data* to stress that we are foremost concerned with visualising instance data and not the »schema« of ontologies[2].

2.2.4 The Semantic Web and its Languages

In this thesis, we focus on ontologies written in the languages standardised by the W3C as part of the Semantic Web technology stack.

The **Resource Description Framework** (RDF) [RDF04a] is the basis of this stack. It allows for describing a graph and offers the principle of defining simple statements as triples consisting of subject, predicate, object. The **RDF Schema** (RDFS) [RDF04b] allows for defining relations (properties in RDFS) and classes for typing the RDF data. The **Web Ontology Language** (OWL) [OWL04] allows for using Description Logics to reason about RDF resources. Following the principle of Description Logics, which allow for describing a specific logic by combining the required operators, also OWL is not one single language, but was specified as a language family in three variants with different expressivity and decidability. *OWL Full* offers the maximum expressivity. *OWL Light* offers decidability, but focuses on a few basic language constructs, which are sufficient to describe taxonomies, for example. *OWL DL* is

[2] Some authors use the term *ontology* only for the T-Box and *instance data* for the A-Box, e. g., in [UG04] where both together are then referred to as a *knowledge base*.

a compromise, which offers reasonable reasoning times for many purposes at the cost of some constraints on expressivity, e. g., relations (properties) may not appear as objects or subjects in a statement. Our approach is specific to OWL and not only RDFS, since with OWL it is possible, for example, to define characteristics of relations such as *symmetry* or *transitivity*.

In the following, we give a brief overview on some important concepts of OWL that we refer to in the next chapters. To describe classes by restrictions, OWL provides:

owl:allValuesFrom is equivalent to the *universal quantifier* (∀) from predicate logic. Restricting a class $C1$ on a property $P1$ to $C2$ using owl:allValuesFrom means that an individual belonging to class $C1$ is any individual that is related by $P1$ only to individuals belonging to $C2$.

owl:someValuesFrom is equivalent to the *existential quantifier* (∃) from predicate logic. Restricting a class $C1$ on a property $P1$ to $C2$ using owl:someValuesFrom means that an individual belonging to class $C1$ is any individual that is related by $P1$ to at least one individual belonging to $C2$.

Furthermore, we sometimes refer to different classes of properties that OWL defines:

owl:ObjectProperty is used to relate individuals.

owl:DataProperty is used to relate individuals to data values, e. g., integers or strings.

SPARQL [SPA08] was developed as a query language for RDF data. Similar to SQL, queries can be formulated to select a tabular result set from an RDF graph (SELECT operation), or to construct a new RDF graph from the data (CONSTRUCT operation). From version 1.1, the SPARQL Query language is accompanied by a set of related recommendations such as the **SPARQL 1.1 Update Language** [SPA13], which provides functionality to delete triples and create new ones. The UPDATE, INSERT and DELETE operations introduced in SPARQL 1.1, expand SPARQL from a query language to a means for manipulating RDF data.

A benefit of focusing on the RDF technical space, is the wide spreading of ontologies written in RDFS and OWL. For many ontologies written in other popular formats – such as OBO [OBO] in the life science domain – translations to OWL exist [GHH$^+$07].

2.2.5 Linked Data and Open Data

While many websites are internally based on structured data sources (e. g., relational databases like MySQL), data is usually only provided as unstructured documents to the users as HTML pages or PDF files. This lack of raw data has been criticised, for example, by Tim Berners Lee [BHBL09], who requests to open up data silos and provide structured data in machine-readable formats.

Linked data makes structured data available for use. The idea is that browsers are enabled to download new information (new RDF triples) on a resource, by dereferencing its URI (Uniform Resource Identifier) and interpreting it as a URL [BHBL09]. The triples downloaded from the URL can point to further URIs, which the browser can follow in turns and so forth and so on, which has been called the »follow-your-nose strategy« (e. g., [Hau09]). First sources of Linked Data were the DBpedia[3], an extract from structured Wikipedia data, and Freebase[4].

The paradigm of **Open Data** aims at data that is (among other criteria) freely accessible [BK12]. Open Data can be stored in relational databases and made available for editing by the crowd. For example, on factual.com[5] tables of Open Data can be edited. This means Open Data does not necessarily be stored as **Linked Open Data**, although the terms are often used in conjunction.

[3] http://www.dbpedia.org; [ABK$^+$07]

[4] Freebase has been announced to be shut down.. http://www.freebase.com, accessed: 02.07.2015.

[5] Factual. http://www.factual.com, accessed: 02.07.2015.

2.2.6 The Metamodelling Technological Space

Having provided some background on the ontology technological space and the technologies of the Semantic Web, we also need to briefly describe its counterpart commonly used in the software modelling domain – the metamodelling technological space. A definition of **metamodel** is given by Seidewitz:

> » A metamodel makes statements about what can be expressed in the valid models of a certain modelling language.
>
> *Ed Seidewitz [Sei03]* «

Beyond this very general definition, *metamodelling* is often associated with the technologies and standards defined by the *Object Management Group*[6] (OMG), such as the **Meta Object Facility**[7] (MOF). MOF is the metadata architecture the OMG recommends for metamodelling. It is a four-layered architecture consisting of four layers: M0 (objects, e. g., the objects in a program written in an object-oriented programming language), M1 (models, e. g., a concrete UML model), M2 (metamodels, e. g., UML), and M3 (the metametamodel). The layer M3 is the last layer, since it can be used to express the MOF in itself. As a means to specify constraints and object query expressions to a MOF model, the *Object Constraint Language*[8] (OCL) is recommended by the OMG.

In this thesis we sometimes refer to Eclipse-specific[9] technologies such as the **Eclipse Modeling Framework**[10] (EMF). The EMF implementation of the MOF is **Ecore**.

2.2.7 SPIN

The *SPARQL Inferencing Notation*[11] (SPIN) [KHI11] is a W3C Member Submission promoted by *TopQuadrant*[12]. It represents SPARQL in RDF, so that it can be stored alongside the data that is queried. SPIN further is a set of RDF vocabularies that allow for defining inference rules and constraints. Defining a constraint on an RDFS class is similar to how constraints can be added to UML classes with OCL. Beyond rules and constraints, SPIN also allows for defining functions, reusing queries by parametrisable templates, and specifying attributes, like in UML. Thereby, SPIN brings the object-oriented paradigm and metamodelling capabilities to (RDF-based) ontologies.

Besides proprietary implementations of SPIN for products of TopQuadrant, an open-source implementation of SPIN, the SPIN API[13], is available. Since constructs such as constraints and rules represent encapsulated SPARQL and SPARQL Update requests, they can internally be processed by a SPARQL engine.

We already refer to SPIN and related technologies such as UISPIN during the analysis of the state of the art (Sect. 4.3.1). Additionally, we discuss SPIN in more detail under various aspects in the remainder of this thesis: In the context of the RVL language (Sect. 7.11), we use the metamodelling capabilities of SPIN. In the context of guidance (Sect. 8.1), we use SPIN for formulating further constraints that are evaluated to guide the visualisation process.

[6] Object Management Group. http://www.omg.org, accessed: 02.011.2015.
[7] Meta Object Facility. http://www.omg.org/mof/, accessed: 02.011.2015.
[8] Object Constraint Language. http://www.omg.org/spec/OCL/, accessed: 03.11.2015.
[9] Eclipse. http://www.eclipse.org, accessed: 02.07.2015.
[10] Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/, accessed: 02.07.2015.
[11] SPIN. http://spinrdf.org/, accessed: 02.07.2015.
[12] TopQuadrant. http://www.topquadrant.com, accessed: 02.11.2015.
[13] SPIN API. http://topbraid.org/spin/api/, accessed: 02.11.2015.

2.3 Guidance

The guidance of the user is a defining requirement for the OGVIC approach; therefore, we briefly give a definition of guidance, discuss different kinds of guidance and show where in the visualisation pipeline the user can be guided.

Definition 5 (Guidance) *The »act or process of guiding« where guiding is defined as to »direct or control the path or course of (something)« [MWa].*

This definition leaves much space on the degree of enforcement of guidance rules (»direct« vs. »control«); therefore, we introduce *levels of guidance* in Chapter 8. Furthermore, according to Si-Said et al. [SSR98], guidance can be categorised into *step guidance* and *flow guidance*. Step guidance concerns guiding a user in fulfilling a single process step. Flow guidance concerns guiding a user from one step in a process to the next step in a process that is the best to perform. Both kinds of guidance are relevant to visualisation processes.

2.3.1 Guidance in Visualisation

Guidance for visualisation has been discussed by Bull [Bul08] and is inherently interlinked with tooling, since it is the system that shall guide the user. Guidance can be offered for all customisation steps in the visualisation reference model (cf. Fig. 2.2), whenever the user interacts with the system. The user can be guided when filtering data, when performing the visual mapping and also when performing view transformations.

Guidance for Filtering When filtering the source data, one means for guiding the user is *faceted browsing*, which became popular in recent years. The user is restricted by construction of the GUI in order to allow only those filtering queries that return results. Ben-Yitzhak et al. [BGH+08] also refer to faceted browsing as *guided navigation*.

Guidance for Visual Mapping When performing the visual mappings the user may be guided to select expressive, effective and appropriate visual means (cf. Sect. 5.7). Some visualisation design systems from the field of statistics analyse the data and data types in order to suggest visual means based on the probable scale of measurement (e. g., nominal, ordinal, quantitative; cf. Sect. 5.6.2).

Guidance for View Transformations Even in the final rendered view, the user may be guided, for example by enabling navigation along visual structures in addition to a free movement through the graphical space.

The focus of the OGVIC approach is on the second case, the guidance for performing the visual mapping. We show that with the emergence of highly structured data, new and different opportunities for guiding the user are possible. These go beyond the abilities of classic visualisation design systems for tabular statistical data.

Chapter 3

Problem Analysis

Data stored in ontologies is highly interrelated and has formal semantics offering good opportunities for connecting various data sources and enabling complex querying and filtering. However, intentionally, this data is completely free from presentation, structuring and formatting information. On the one hand, this means that the pure ontological data is hard to read and understand by humans. On the other hand, this can be seen as an advantage, since the raw shape of data offers ideal conditions for visualisation. We argue that current ontology visualisation approaches often do not exploit the benefits that ontological data offers. Furthermore, ontologies could help to overcome problems with the interoperability of visualisation systems and the exchange of visualisation knowledge.

In this chapter, we list the problems we identified in this context (Sect. 3.1). Based on these problems, we formulate concretised research questions (Sect. 3.2) and describe the case studies we conducted (Sect. 3.3) in order to analyse concrete ontologies from various domains (Sect. 3.4) and identify frequently occurring visual mapping situations (Sect. 3.5 and 3.6). This problem analysis chapter results in a list of precise requirements for our ontology-driven visualisation approach (Sect. 3.7).

When identifying concrete problems in the next step, in many cases we describe these problems from the perspective of a specific actor. Therefore, we briefly introduce these actors and their use cases. Both actors are typically experts on their domain (e. g., biotechnology), but not necessarily skilled in visualisation or programming:

- The **Visualisation Author** – Her goal is to visualise ontological data that she found on the Semantic Web or that is available from local ontologies. After specifying a set of helpful visual mappings, she wants to send her visualisation settings to a colleague who told her that he needs to visualise a similar ontology. In a later project, she also needs to reuse the settings herself. However, some changes and extensions have to be done, since the ontology grew in the meantime. Additionally, her company decided to use a completely different visualisation suite. A third colleague also defined some visualisation settings. She could save some time if it were somehow possible to combine his settings with hers.
- The **Domain Author** – She created a new domain ontology and wants to suggest a good visualisation that is helpful for people using her ontology. Just like authors of XML data sometimes offer *style* sheets, she would like to offer recommendations for *visualising* her ontology.

3.1 Problems of Ontology Visualisation Approaches

We identified the following eight problems of current visualisation approaches for ontological data[1]:

P-1 Often no visual mapping exists, but a general transformation of data to document shape

P-2 Visualisations often follow a single visual paradigm (e. g., Node-Link-Diagrams), while others are ignored

To make use of the human eye's capability of quickly perceiving complex data sets, we need to encode data variables to visual means. It is not sufficient to bring a huge ontology file into a document shape *(P-1)* or simply reflect the graph structure of the ontology by using a trivial node-link-diagram representation *(P-2)*. The ontological data from our case studies greatly varies with respect to its scale of measurement, data types and structure. Since different visual structures have different effectiveness for encoding different kinds of data, multiple visual structures should be provided to create an appropriate tailor-made visualisation instead of using generic visualisations.

P-3 Visualisation authors cannot reuse visual mappings in other tools and share them with other people

P-4 Visualisation authors cannot compose their own mappings with existing ones

P-5 Domain authors do not have a standard format to ship visualisation information along with their domain ontologies

Ontology visualisations are created for specific platforms and separately for each visual paradigm. They cannot be shared with others and reused on other visualisation platforms *(P-3)* or in combination with different visual paradigms *(P-4)*. Whereas more and more standardised formats are used for storing domain knowledge, visualisation settings are not stored based on standards *(P-5)*.

P-6 A visualisation language for ontological data does not exist

Even though not standardized, a few general visualisation languages exist (Sect. 4.2.2). However, these cannot directly reference ontology specific constructs such as individuals, classes and relations *(P-6)*. As an example, visualisation authors need to be able to conveniently reference relations between classes that exist indirectly by existential or universal restrictions.

P-7 Generic visualisation design systems do not exploit the specifics of ontological data

P-8 Visualisation authors cannot exchange their system's expert knowledge

Currently, only for tabular, statistical data good visualisation design systems exist (Sect. 4.1.1) that offer basic guidance functionality for simple data types based on internal visualisation knowledge and the analysis of the data. However, they cannot guide the user, when it comes to visualising ontological data with its specifics *(P-7)*. As a basis to reduce visual mapping options to a useful subset, a visualisation approach should consider all the information it can extract from the ontology. Examples are inverse relations or the symmetry and transitivity of relations as well as subproperty relationships. An additional prerequisite for guiding the visualisation author is expert knowledge on visualisation. This visualisation knowledge is usually hard-coded into the system and cannot easily be exchanged, when new empirical results are available *(P-8)*.

[1] While we concentrate on the visualisation of ontological data in this thesis, we are aware that many of the discussed problems, just like many of our proposed solutions, could also be applied to general visualisation systems.

3.2 Research Questions

Our main research question is: *How can we support the tailor-made, effective and reusable visualisation of ontological data?* Starting from this very general question and taking into account the problems identified in the last section, we formulate the following two concretised subquestions, which in turn have two subquestions each:

Q-1 How can we define composable and shareable mappings from ontological data to visual means? (*P-1*, *P-2*, *P-3*, *P-4*, *P-5*, *P-6*)

Q-1.1 Which ontology constructs do we want to map and onto which visual means? (*P-2*, *P-6*)

Q-1.2 What should be the target model we map to? (*P-2*, *P-3*, *P-4*, *P-5*)

The first group of questions above aims at the general problems of reusing and composing visual mappings, while the group below aims at the question of how end-users can be supported in creating these mappings.

Q-2 How can we guide the visual mapping of ontological data? (*P-7*, *P-8*)

Q-2.1 Can visualisation benefit from the rich semantics of ontological data? (*P-2*, *P-7*) Can it, for example, help to allow for other visual paradigms than the node-link paradigm?

Q-2.2 How can we formalise expert visualisation knowledge? (*P-8*)

Although we split our research questions into these two groups, almost none of the questions can be thought separately from the others; therefore, all six questions are subject to this thesis. In the remainder of this thesis, we try to answers these six research questions starting with Question *Q-1.1*, which is answered already in the following sections of the Problem Analysis.

3.3 Set up of the Case Studies

In order to approach research question *Q-1.1*, case studies have been conducted for eight ontologies from three different domains – *publishing, life sciences* and *software requirements*. This included:

1. Analysing the case studies ontologies by identifying ontological concepts such as *transitivity*, *inverse relations*, *subproperties* and *domain-range relations* and counting their occurrences
2. Selecting ontological concepts according to the frequency of their usage but also to cover a broad range of different ontological concepts
3. Interviewing domain experts and doing a (subjective) review of domain literature in order to identify common visualisations from the respective domain
4. Drawing manual sketches for selected concepts
5. Analysing these sketches for visualisation cases
6. Testing the prototypical visualisation system with respect to the manual sketches

While it was not possible to perform a detailed analysis of the graphics used in arbitrary domains, we see this procedure as a compromise of effort and generality, which allows for working with real world examples from three very different scenarios. As opposed to an analysis that only focuses on the popularity or usefulness of visualisations in the respective domains, our approach enabled us to consider as well, which ontologies are actually available and which ontological power they have.

In the following, we briefly introduce the examined ontologies for each of our three case study domains. Afterwards we describe each of the steps listed above in more detail, except for the last step (testing), which is described at the end of this thesis (Chapter 9).

3.3.1 Case Studies in the Life Sciences Domain

In the life sciences domain, we closely inspected three ontologies from biology and biotechnology. The well-known Gene Ontology was intentionally excluded, since it uses only two different ontological relations – *is a* and *part of*.

- The Plant Ontology (PO)
  http://purl.org/obo/obo-all/po_anatomy/po_anatomy.owl
  http://purl.org/obo/owl/po_temporal
- The Zebrafish Anatomy Ontology (ZFA)
  http://purl.org/obo/owl/zebrafish_anatomy
- The Amino Acids Ontology[2]
  http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl

The **Plant Ontology** is an OBO[3] ontology. Parts of it have been transformed to various OWL ontologies such as the plant anatomy and the temporal module that we chose for our case study. The anatomy module mainly represents a obo:part_of-hierarchy of plant components, the temporal module offers information on plant development stages (obo:develops_from). The high number of change requests[4] suggests that the Plant Ontology has an active user community.

The **Zebra Fish Anatomy Ontology** is likewise a transcript of an OBO ontology to OWL. The ontology represents a obo:part_of-hierarchy of the fish sections, and the sequence of development stages (obo:develops_from). Additionally, obo:start and obo:end define the duration of each stage.

The **Amino Acids Ontology** was developed by Robert Stevens and Phillip Lord at the University of Manchester and »*combines many aspects of the physicochemical properties Taylor [Tay86] uses to classify amino acids to give a rich, multi axial classification of amino acids*«[5].

Rather than instance data, the Plant Ontology, the Zebra Fish Anatomy Ontology and the Amino Acids Ontology represent knowledge in the T-Box as existential constraints (cf. Sect. 2.2.3). Besides this, in the Amino-Acids ontology, properties are described via domain–range axioms. All three ontologies contain rdfs:subClassOf-hierarchies and use various owl:AnnotationPropertys such as oboInOwl:hasSynonym.

3.3.2 Case Studies in the Publishing Domain

In the publishing domain, we closely inspected two ontologies:

- The Citation Typing Ontology (CiTO)
  http://purl.org/spar/cito/
- The Bibliographic Ontology (BIBO)
  http://purl.org/ontology/bibo/

CiTO is part of a collection of ontologies from the publishing domain called SPAR[6]. It consists of eight main ontologies, whereof we have a closer look at CiTO, the Citation Typing Ontology and FaBiO, the FRBR-aligned Bibliographic Ontology. CiTO is already used in opencitations.net[7]

[2] Amino Acids Ontology (download). http://www.cs.man.ac.uk/~stevensr/ontology/amino-acid.owl, accessed: 09.11.2015.
[3] The Open Biological and Biomedical Ontologies. http://obofoundry.org/, accessed: 09.11.2015.
[4] Plant Ontology (issue tracker). https://github.com/Planteome/plant-ontology/issues/, accessed: 09.11.2015.
[5] From the abstract of the Amino Acids Ontology
[6] SPAR Ontologies. http://purl.org/spar/, accessed: 02.07.2015.
[7] Open Citations. http://opencitations.net/, accessed: 18.12.2014.

and citeulike[8]. CiTO offers a large number of different object properties that allow for describing various ways of how one document cites another (e. g., *DocumentA* cito:disagreesWith *DocumentB*). With respect to visualisation, this raises the question of how to support the visual distinction of theses relationships.

BIBO is in competition with SPAR. Unlike SPAR, BIBO defines all of its terms in a single ontology and has a narrower domain. It focuses on bibliographic entities, but also offers some terms of other fields of publishing, such as a citation relationship (without subtypes) and document identifiers such as DOI and ISBN. Unlike SPAR, it supports lists of authors. Although not being as detailed and comprehensive as SPAR, it seems to be more stable for the time being and is already reused in many other ontologies such as VIVO[9].

For our case study we instantiated CiTO and BIBO with a fictional set of six bibo:Documents citing each other by various subproperties of cito:cites.

3.3.3 Case Studies in the Software Technology Domain

In the domain of software technology we inspected the **Requirements Ontology**, which offers a vocabulary to formally define requirements, goals and use cases. The ontology has been developed by Siegemund et al. and was used in the context of ontology-driven requirements engineering [STZ+11].

- Requirements Ontology (RO)
  http://purl.org/ro/ont

In our case study the Requirements Ontology is instantiated by an example scenario that describes requirements for a social network application. This example data was provided by the authors of the ontology. Modelled relations include ro:refinesTo (for refining goals) and ro:hasCost to state the costs of a requirement. Some characteristics are expressed by assigning additional types, e. g., ro:HighPriority is modelled as a class.

3.4 Analysis of the Case Studies' Ontologies

The analysis of the case studies' ontologies first of all included the identification of ontological concepts[10] that could be candidates for mappings to visual means.

Which ontological concepts and relations may be visualised? Fig. 3.1 gives an overview of ontological resources that can be addressed when creating visualisations. As an example scenario, we show part of a fictional knowledge base about foaf:Persons that are the dc:creators of bibo:Documents. The first thing we note is that classes (e. g., foaf:Person), individuals (ex:BookA) and relations (cito:cites) should be directly referencable by a visual mapping vocabulary or system. A second general observation is that individuals and their relations – the A-Box – may be visualised (e. g., we may want to graphically represent that ex:AuthorA is the dc:creator of ex:BookA), but also the relations between types – the T-Box – may be subject to visualisation. For example, often classes defined via property restrictions form complex graphs that could be visualised. In Fig. 3.1, we give a brief example of an existential restriction on the class bibo:Book, for which it is stated that there exists some foaf:Person. However, longer chains of such indirectly stated T-Box relationships were frequently found in our use case ontologies, e. g., in the Plant Ontology. Another indirect relationship between classes is sometimes given by domain-range settings (in our example, we show domain and range settings for the ex:childOf relation).

[8] Blog post on citeulike. http://opencitations.wordpress.com/2010/10/21/use-of-cito-in-citeulike/, accessed: 02.07.2015.

[9] VIVO describes networks of scientists. http://vivoweb.org/, accessed: 10.10.2015.

[10] See Sect. 2.2.3 for a brief introduction to ontological concepts

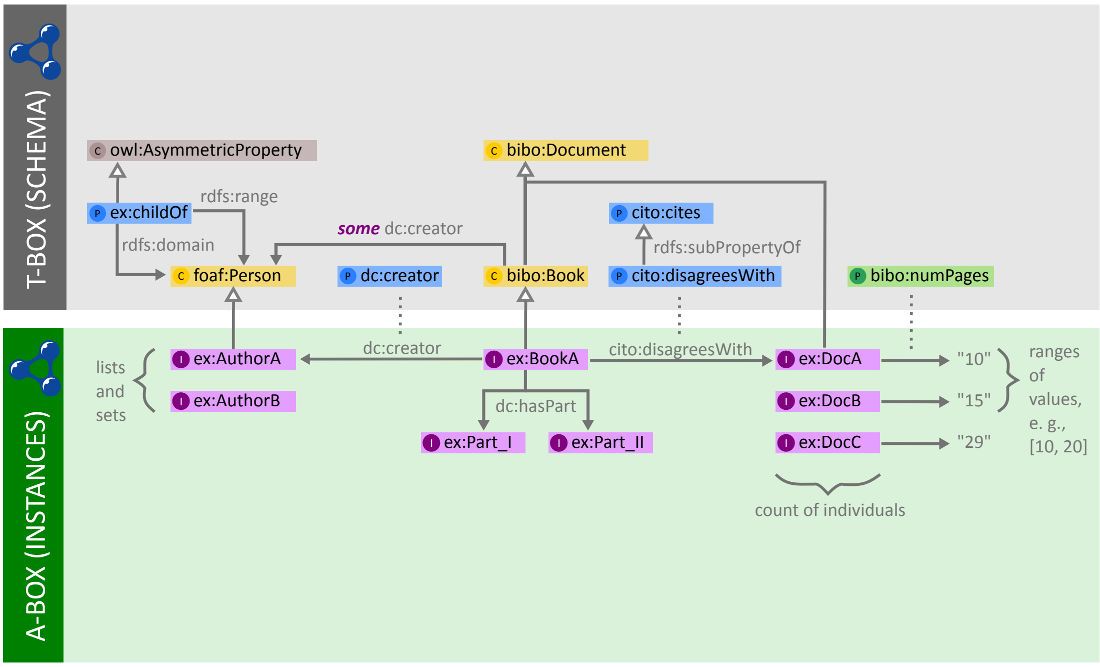


Figure 3.1: Identified ontological concepts and relations of the A-Box (bottom pane) and T-Box (top pane) which could be mapped to visual means. The identified concepts and relations are illustrated using a fictional knowledge base on persons and documents.

Third, it is important to distinguish »attributes« from »relations«: owl:ObjectProperty is only sometimes used to describe relations between complex objects, while there are also cases, in which owl:ObjectPropertys may point to simple data (strings, integers) »packed« into a resource to allow for referencing it. This means, when we are looking for attributes in our data that could be mapped, we have to consider properties of all types, not only owl:DatatypePropertys. In the following, to abstract from the concrete property type used on the RDFS/OWL modelling level, we simply differentiate between *attributes* (often, but not always typed as owl:DatatypeProperty) and *relations* (often, but not always typed as owl:ObjectProperty).

Further, in the A-Box of our example knowledge base, we may want to reference sets and lists of values (e. g., a list of authors) or count individuals (e. g., all resources of a given type that can be found in the database). Especially, when referring to values of simple data types (e. g., bibo:numPages), we may want to select ranges (intervals) of values rather than selecting single specific values. Similarly, we sometimes may need to select only a subset of resources for visualisation purposes, even when there is no asserted or defined class in the ontology that describes this subset (e. g., all bibo:Documents with at least two subparts).

Taking a closer look at the T-Box, we find that besides the relations (properties in RDFS) themselves, also the *characteristics of relations* could be used in visualisation settings (e. g., to specify that an owl:AsymmetricObjectProperty should be visualised as directed line-connectors with arrowheads and owl:SymmetricObjectProperty only as undirected line-connectors).

Related to the question of what needs to be selectable for visualisation is the question, what should not be required to be explicitly selected. For example, often subproperties are used to relate instances. In this case, we might not want to define a visualisation for each of the subproperties, but expect that mapping a superproperty will be sufficient. A mapping of cito:cites should – by default – also apply to statements using the subproperty cito:disagreesWith. At the same time, there may be situations, where we need to easily define a different visual mapping for all subproperties of cito:cites.

Often	Sometimes	Rarely
Transitivity	Functional	Asymmetry
Reflexivity	Symmetry	Irreflexivity
Inverse	Anti-Symmetry	Disjointness
		Inverse functional
		Cardinality constraints

Table 3.1: Characteristics of relations ordered by frequency of occurrence in the case studies' ontologies.

Which ontological concepts and relations should we focus on? Resources in an ontology are often connected to other complex entities, which may, in turn, have multiple attributes and relations to other entities themselves. This is in contrast to the large homogeneous tables of statistical data, as they emerge from measurements that return values with simple data types such as floats, integers, Booleans or strings. This statistical data, which has been processed already in many existing visualisation approaches, is not typically stored in ontologies. Therefore, we focus on the relations between complex objects and the characteristics of relations that can be specified with OWL. To find the most important ontological concepts and relations, we first examined the availability of these concepts in our test set by counting the frequency of their occurrence. Second, we ordered the ontological concepts according to their expected usefulness for driving visualisation aspects. Table 3.1 shows an example of this ranking for OWL relation characteristics.

3.5 Manual Sketching of Graphics

To identify the graphics that are common in the case studies' domains, a (subjective) literature review and interviews with domain experts were done. However, our hypothesis was that the availability of ontological data with formal semantics allows for new visualisation possibilities and different, richer graphics than currently used in many domains. Therefore, when sketching a set of graphics that represent ontological data from a certain domain, rebuilding typical graphics that are commonly used in a field was only one goal. Additionally, we inspected existing ontology visualisations and tried to create alternative graphic representations, aiming at both a better exploitation of the available semantics and a better usage of the full palette of visual means. The graphics were first sketched manually and only later refined with a vector graphic program to avoid constraining the visual possibilities by technical burdens. In the next subsection, we present and analyse a subset of these graphics[11].

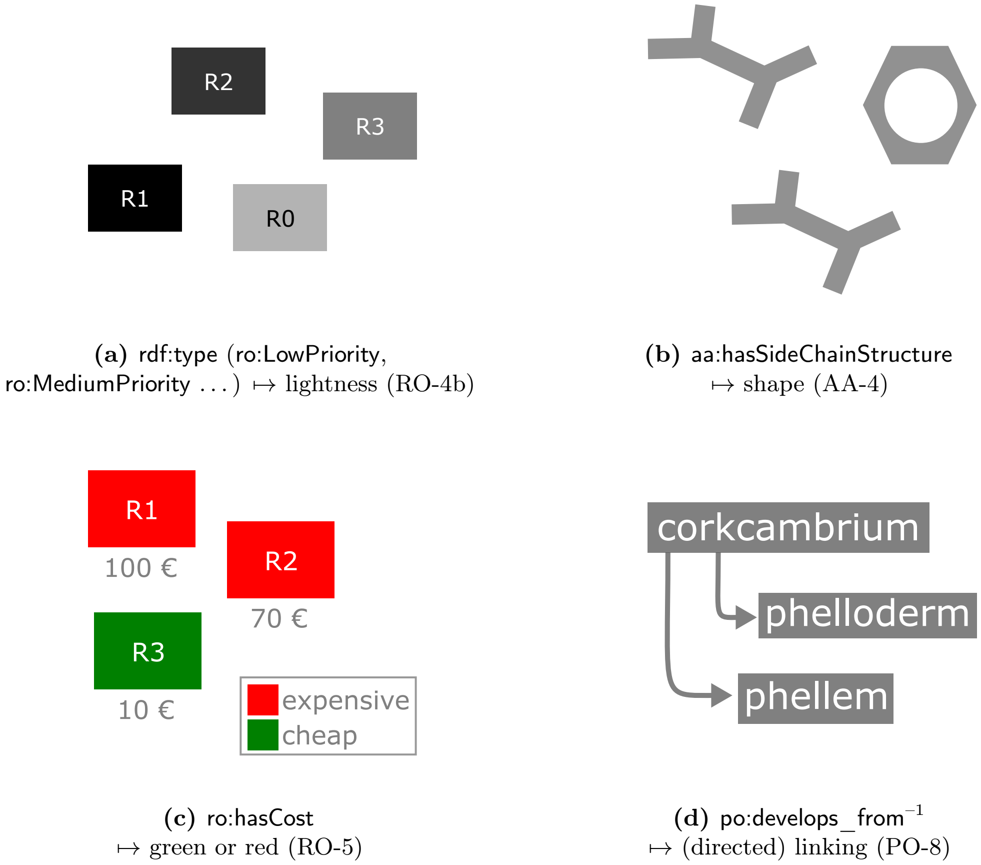
3.6 Analysis of the Graphics for Typical Visualisation Cases

Figures 3.2, 3.3 and 3.4 show a small excerpt of the sketches we created during our case studies. In the following, at the example of this excerpt, we describe 12 Visualisation Cases (VC) identified during the analysis of these sketches. The VC include typical cases of defining and combining elementary visual mappings, but also other cases of specifying visualisation related settings.

The identified VC add to the requirements of our visualisation approach (Sect. 3.7) and especially to the requirements we set up for the design of an RDFS/OWL visualisation language in Chapter 7.

VC-1 Create a graphic object per resource. Often graphic objects need to be created for the main resources of interest that shall be visualised. These graphic objects can then

[11] A complete list of the sketches that were done for the case studies' ontologies can be found in Appendix A.



(a) rdf:type (ro:LowPriority, ro:MediumPriority . . .) ↦ lightness (RO-4b)

(b) aa:hasSideChainStructure ↦ shape (AA-4)

(c) ro:hasCost ↦ green or red (RO-5)

(d) po:develops_from^{-1} ↦ (directed) linking (PO-8)

Figure 3.2: First set of sketches for the case studies' ontologies *Requirements Ont.* (ro:), *Amino Acid Ont.* (aa:), and *Plant Ont.* (po:). The interval labels »expensive« and »cheap« used in the legend of subfigure (c) are not defined in the Requirements Ontology and will have to be added during the visualisation process.

be assigned graphic attributes and may be graphically related to other objects. Other resources of secondary interest, which are not in focus of the graphic, are often encoded into graphic attributes of existing graphic objects rather than into new graphic objects. For example, in Fig. 3.2a the priority of a requirement is not represented by its own object, but only by the lightness of the graphic object representing a requirement.

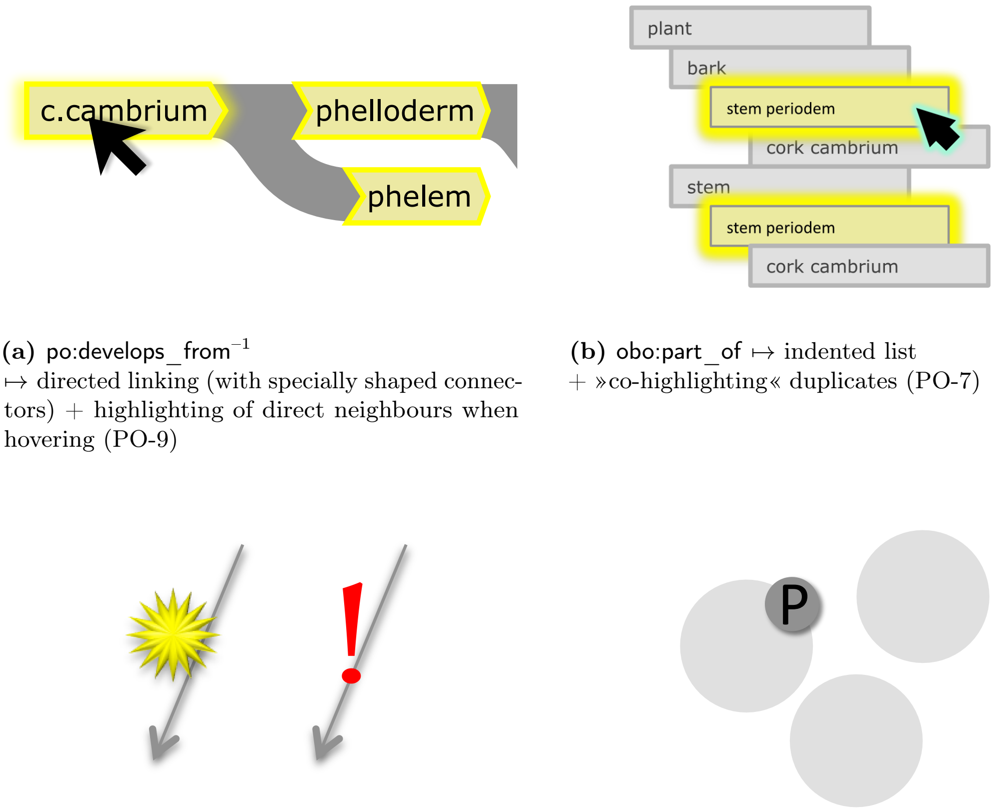
VC-2 Map to Graphic Attributes. Mapping a property of a resource to a graphic attribute is a basic visualisation case. A second example for it can be found in Fig. 3.2b, where shape is used to differentiate subtypes of Amino Acid. However, a 1-to-1 mapping of values is not sufficient, neither is it possible to map values automatically in all cases. As an example, as Fig. 3.2c illustrates, we sometimes need options for manually mapping ranges of source values.

VC-3 Map to Graphic-Object-to-Object-Relations. Most of the sketches do not only make use of graphic attributes, but use various other (spatial) relations that can exist between graphic objects and that we refer to as *Graphic-Object-to-Object-Relations* [vE02] in this thesis. These include the frequently needed *linking* (Fig. 3.2d) and *containment* (Fig. 3.4b), but also other relations are used: For example, in Fig. 3.3b alignment is used to represent equal levels in a (part-of) hierarchy, which results in the common representation type of an *indented list*.

VC-4 Create additional graphic objects. While some resources are not represented by their own graphic object, there are also cases, where additional graphic objects need to be created, for example, in order to represent a relation to another resource or an attribute explicitly via a connector object (Fig. 3.2d) or to represent an attribute via a label object (Figures 3.3c, 3.3d).

VC-5 Define simple interactions. In many situations simple interactions can be used such as highlighting objects based on their relation to a second object, which the user selects or moves the mouse pointer over. A frequently occurring case is that of duplicated graphic objects both representing the same resource (Fig. 3.3b). Duplicating graphic objects becomes necessary every time a graph needs to be turned into a tree structure for presentation. Another case for highlighting is that of related neighbours (Fig. 3.3a).

VC-6 Simplify the ontological model. While not actually being a matter of visual mapping, it is often necessary to »prepare« the source data RDF graph, before mapping it to visual means. In this step, information can be simplified, e. g., from two inverse properties, one can be picked as the preferred one, while hiding the other from the graphic. Similarly, for transitive properties, derivable statements may be undesired in the graphic and need to be hidden as well.



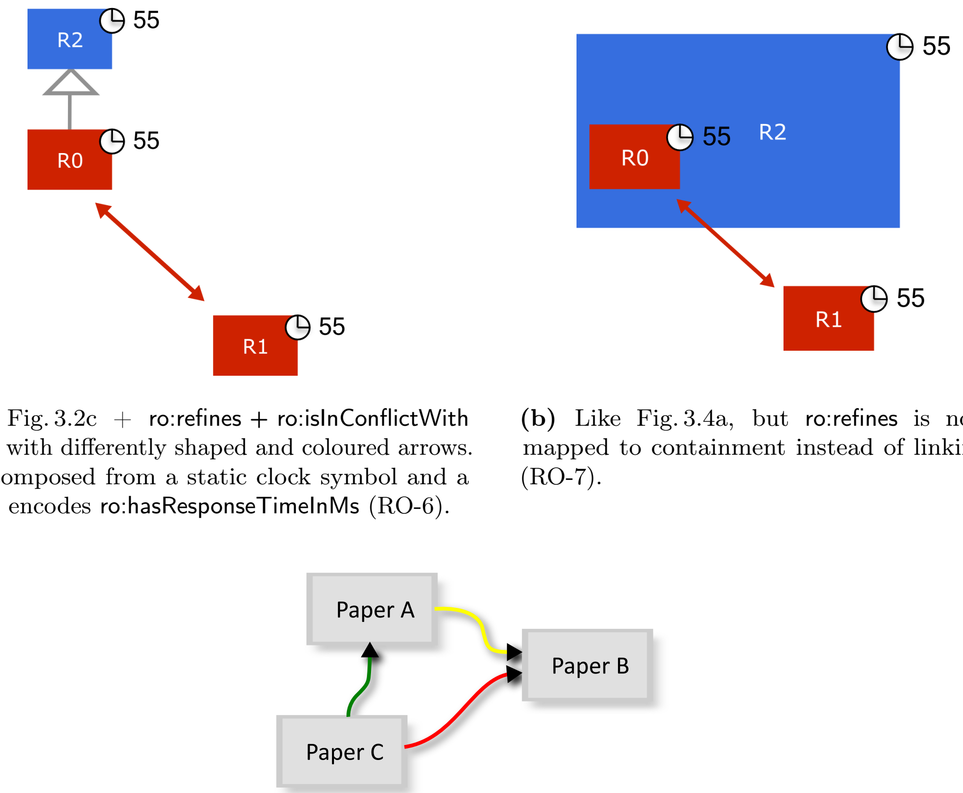
(a) po:develops_from^{-1}
↦ directed linking (with specially shaped connectors) + highlighting of direct neighbours when hovering (PO-9)

(b) obo:part_of ↦ indented list
+ »co-highlighting« duplicates (PO-7)

(c) cito:cites
↦ directed linking + iconic labels on the connector to distinguish subproperties of cites (CIT-5)

(d) aa:hasPolarity
↦ label with a »circled P« shape but only if the value is aa:Polar (AA-3)

Figure 3.3: Second set of sketches for the case studies' ontologies *Plant Ont.* (po:), *Citation Ont.* (cito:), and *Amino Acid Ont.* (aa:).



(a) Like Fig. 3.2c + ro:refines + ro:isInConflictWith ↦ linking with differently shaped and coloured arrows. A label, composed from a static clock symbol and a text value encodes ro:hasResponseTimeInMs (RO-6).

(b) Like Fig. 3.4a, but ro:refines is now mapped to containment instead of linking (RO-7).

(c) cito:cites ↦ directed linking + subproperties distinguished by a set of ordered colours (traffic light set) (CIT-1).

Figure 3.4: Third set of sketches for the case studies' ontologies *Requirements Ont.* (ro:) and *Citation Ont.* (cito:).

VC-7 Reuse/extend/compose mappings. Some of the mapping cases can be found many times in the case studies' sketches. For example, Fig. 3.4a and Fig. 3.4b share multiple mapping settings, e. g., colour is used to encode the costs of requirements in both graphics. Only the refinement relation between requirements is mapped differently in the two graphics – while Fig. 3.4a uses linking (with a UML-like arrow head), Fig. 3.4b uses containment instead. It should be possible to extend the settings made for Fig. 3.4a and only add the mapping to containment. Furthermore, it should also be possible to reuse two already defined mappings and compose them to an new more complex graphic as it was done in Fig. 3.5.

VC-8 Use complex standard graphics. Complex graphic representations such as tree maps and tables with column and row labelling should be easy to define. Nevertheless, it should be possible, to add further mappings to such representations, as in Fig. 3.5.

VC-9 Refer to parts of the graphic. When multiple mappings are applied in combination, later mappings may refer to specific graphic objects created by earlier mappings. For example, in Fig. 3.4c, the mapping on colour is applied to the *connector* introduced by another mapping.

VC-10 Draw legends and labelled axes. While some visual mappings can be understood intuitively by the human observer based on experiences with conventions or the physical environment (e. g., blue for cold/ice, red for fire/heat), in many situations (e. g., Fig. 3.2c) legends are necessary to allow for a precise visual »decoding« of mappings.

VC-11 Define styles. In the sketches, some graphic settings exist that do *not* encode information and which cannot be controlled by mappings. Examples of such settings that we call *styles*, are the border- and background colours of objects (Fig. 3.3b) and shadows (Fig. 3.4c). Styles assign a single constant value or a combination of constant values to a set of elements. One of multiple possible values for a graphic attribute is chosen, without necessarily encoding meaning.

VC-12 Benefit from good defaults. Style settings should be based on good defaults. An example for this is the colour of text (dark coloured font on bright backgrounds, bright on dark ones; Fig. 3.2a) or the concrete appearance of directed connectors (e. g., an arrow with a middle large arrow head; Fig. 3.2d). Beyond style settings, also other defaults could be assumed for many sketches. For example, the labelling of graphic objects with the label of the resource they represent is such a setting that could become a mapping default.

3.7 Requirements

Based on our general goal of effective, tailor-made and reusable visualisations of ontological data by end-users, the concrete visualisation cases we identified in the previous section, as well as on experiences other researchers described for visualisation and presentation approaches (Chapter 4), we formulated 15 requirements (*R-1* – *R-15*) for our visualisation approach. These requirements partly correspond to the criteria for characterising related work, which are described in Sect. 4.1.2 in more detail. In brackets after each requirement, we point to problems (*P-X*), visualisation cases (*VC-X*) and other, more general requirements that explain why we laid down the requirement.

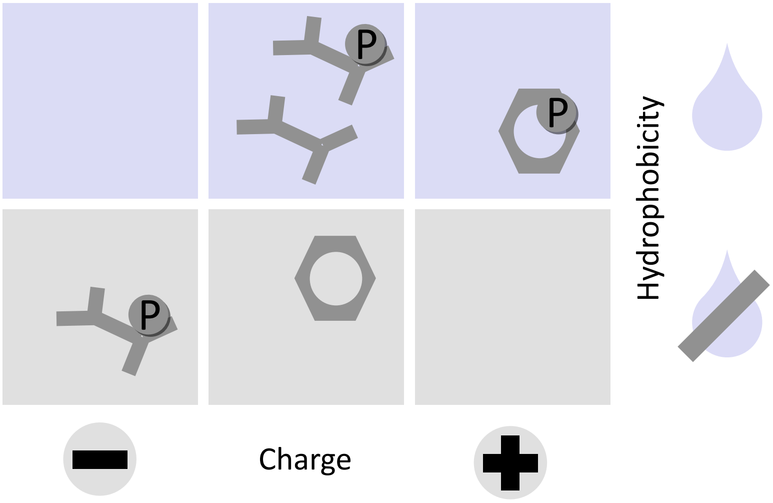


Figure 3.5: Sketch of a tabular representation of amino acids from the *Amino Acid Ont.* (aa:) using two spatial dimensions to segregate amino acids by aa:hasCharge and aa:hasHydrophobicity. Additionally, mappings from aa:hasSideChainStructure and aa:hasPolarity to shape (Fig. 3.2b) and labelling (Fig. 3.3d) are reused (AA-5).

3.7.1 Requirements for Visualisation and Interaction

R-1 Dynamic and value-dependent visual mapping – not only presentation (*P-1*, *VC-2*, *VC-3*)

R-2 Variety of graphic relations (*P-2*, *VC-3*)

R-3 Interaction with the visualisation (*VC-5*)

First, we require our approach to support configuration of **dynamic and value-dependent visual mappings** *(R-1)*, i. e., to allow for actual *visualisation* as opposed to just structuring data into a document shape or formatting data (font type or background colour). Further, instead of focusing on node-link diagrams (graphs) alone, the approach shall cover a **large variety of graphic relations** *(R-2)* such as containment (enclosure), alignment and the separation of graphic objects by a separator. Additionally, also **simple interactions** *(R-3)* need to be possible to define such as »Highlight all linked nodes on select«.

3.7.2 Requirements for Data Awareness

R-4 Ontology-aware (A/T-Box) (*P-7*, *VC-12*)

R-4a Terminological ontology relations (T-Box) supported (*P-7*)

R-5 RDF supported (*P-7*, *P-8*)

R-6 Domain agnostic (*P-7*)

We require the visualisation approach to be **aware of the specifics of ontologies** *(R-4)*. That means processing and representing RDF on a graph level is not considered sufficient and implies that we need to allow for referencing ontology terms within visualisation settings. We further require that also a **specific visualisation of T-Box statements** is possible and that these relations can explicitly be referenced in mappings *(R-4a)*. Since all domain ontologies from our case studies are available in RDF, we add the requirement that **RDF-based ontologies need to be supported** *(R-5)*. Additionally, because we did not restrict ourselves to a certain domain we want to visualise, it is important that our approach is **domain agnostic** *(R-6)*.

3.7.3 Requirements for Reuse and Composition

R-7 Reusability of the defined mappings (*P-3*, *VC-7*)

R-8 Composability of the defined mappings (*P-4*, *VC-7*, *VC-8*)

R-9 Explicit mapping definitions (*VC-7*, *R-7*, *R-8*)

We require visualisation settings made for one data set to be **reusable** for other similar data sets using the same ontological terms *(R-7)*. Visualisation authors should be able to adapt existing settings found on the web to their specific use cases. Related to this, a **composition of multiple visualisation settings** has to be supported *(R-8)* – as far as this is possible with respect to the constraints and rules that apply to the composition of graphics, such as syntactic and perceptual interactions, expressiveness and effectiveness[12]. We require the general composability of visual mappings and demand that our approach offers the foundation for checking constraints and rules of graphic composition. The problem of actually checking the constraints is something that we do not expect to be completely solved in the scope of this thesis. Our definition of composability includes that we allow for mixing visual paradigms such as node-link and tabular representation. Composability of mappings adds to reusability, since only if users can combine mappings with other existing mappings, they can fully benefit from existing work. A prerequisite for composing and reusing existing mappings is that these mappings have to be made **explicit** and can be stored and referenced by name *(R-9)*.

3.7.4 Requirements for Variability

R-10 Platform variability (*P-3*, *P-5*)

R-11 Visual structure variability (*P-2*)

Since we aim at reusing mappings defined by others, e. g., in a different visualisation tool, we require **platform variability** *(R-10)*, i. e., it should be possible to vary the graphic platform (e. g., exchange SVG by X3D) and still use the same visualisation settings. Similarly, mappings should be composable and exchangeable independently as far as possible, even when we **vary the visual structure or visual paradigm**[13], e. g., from list to tree structure *(R-11)*.

We already anticipated that we decided to use a **declarative** approach for defining visual mappings. This was done for reasons of separation of concerns (separating presentation logic from data) and the resulting benefits for reusability [Lie05, Huy07, HB10]. We discuss the advantages and disadvantages of a declarative approach in more detail in the context of the RDFS/OWL Visualisation Language (RVL) in Sect. 7.2.

3.7.5 Requirements for Tooling Support and Guidance

R-12 Domain experts can visualise their data without programming or visualisation skills (*P-7*, *VC-12*)

R-13 Visualisation settings configurable with a GUI (*P-7*, *R-12*)

R-14 Interactions configurable with a GUI (*P-7*, *R-12*)

R-15 Guidance for Visual Mapping with a GUI (*P-7*, *R-12*)

R-16 Consider complex semantics of an ontology for visual mapping (*P-7*, *R-12*)

As described in the introduction to this chapter, our target user group includes **domain experts**, which may lack programming skills or visualisation expertise or both. Since domain experts should equally be able to visualise their data *(R-12)*, a **GUI** should be offered **for the configuration** of visual mapping *(R-13)* and interaction settings *(R-14)*. To allow not only for configuring some visualisation, but support effective visualisations (e. g., adhering to laws of visual perception), we further require **guidance for the visual mapping process via a GUI**.

[12] We come back to rules for graphic composition in Sect. 5.7.3.

[13] We use the term *visual paradigm* for the dominant visual structure in a graphic.

Beyond the guidance support that existing visualisation design suites offer, we require guidance to **consider the complex semantics of ontologies** for the visual mapping advices *(R-16)*.

3.7.6 Optional Features and Limitations

O-1 Configuration results instantly shown

O-2 Data Filtering

O-3 Guidance for Data Selection

O-4 Guidance for View Transformations

L-1 No support for editing the visualised data

The features *O-1* – *O-5* are considered optional with respect to the prototype that has to be built, since they are not in the focus of this thesis. Nevertheless, integrating features such as a mechanism for **data filtering** *(O-2)* is essential for a real world scenario. Technologies for creating visual queries or faceted browsing [ODD06] are possible candidates for filtering. Another problem we do not try to solve with this thesis is to **instantly show the result of configuration changes** *(O-1)*. While the classical visualisation pipeline model/architecture fails with respect to this feature, the need for instantly reflecting changes has been recognized and approached by [Bul08] and also in the context of visual analytics, which requires fast configuration-feedback cycles [KS12].

Two other optional features concern the extension of guidance to other parts of the visualisation process. While we focus on guidance for configuring visual mappings, guidance could be extended to data selection steps (**guidance for data selection**, *O-3*) and to the navigation in the rendered scene (**guidance for view transformations**, *O-4*).

Finally, although we design the visualisation language to be well usable by tooling in order to allow for convenient editing of the visual mapping definitions, **we do not aim at editing support for the underlying source data** *(L-1)*.

Chapter 4

Analysis of the State of the Art

In this chapter we give a detailed analysis of the state of the art in the fields that are touched by this thesis. These fields are approaches to visualisation design, approaches to RDF presentation, the corresponding languages that are used within these approaches, and, finally, approaches to generating user interfaces from models.

Accordingly, the analysis of the state of the art is split into four parts: In a first section, we compare visualisation approaches as a whole, before looking at the languages, that often correspond to one of these approaches, in detail. The comparison of languages is split again into *visualisation languages* and *RDF presentation languages* presented in the second and third section of this chapter. For each of these three areas, we first give a brief overview of each system or language we inspected. Then, we precisely define a certain reference criterion and finally compare all items by this criterion, pointing to the most important differences. A tabular overview sums up and completes the comparison. Finally, in a fourth and last part of this state of the art analysis, we need to review approaches of generating interfaces from models. This is due to the fact that we aim at dynamically generating the visualisation system based on several models – the source data ontologies, visualisation ontologies and a concise schema of a visual mapping language (which will be introduced in the following chapters). Since we already know that we have to reference ontology terms in many situations, we also examine what options are at hand for using »traditional« modelling technologies and ontologies in conjunction.

At the end of each section, we briefly conclude the results of the analysis. We see that none of the visualisation approaches examined here fulfills all the requirements put for our approach; neither does an RDF visualisation language exist, which could be reused.

4.1 Related Visualisation Approaches

In a first step, we analyse visualisation approaches as a whole. Among those are established visualisation design systems from the field of statistics, but also academic systems and prototypes for visualising (RDF) graphs. When choosing the candidates for this comparison, we focused on those approaches, actually being able to perform multiple visual mappings. Therefore, many existing ontology visualisation tools, which use a node-link structure as the only visual means, are not part of this comparison.

4.1.1 Short Overview of the Approaches

First, we briefly introduce each visualisation approach we examined. In Sect. 4.1.2, we then compare all of them in detail by several criteria.

Microsoft Excel

The focus of Microsoft Excel is on data calculation. It can be considered a foremost manual visualisation tool. Although the software offers a wide range of diagram types and shall be mentioned here due to its popularity, MS Excel does not guide the user when choosing one of these types. Also, it is not possible to state the scale of measurement for a variable. Since these are features that Tableau and SPSS Viz Designer add, we rather have a closer look at these products in the following sections.

Polaris and Tableau

Tableau is a comprehensive tool for designing visualisations from tabular data. It is a commercialisation of Polaris, described in [STH02].

The philosophy of Polaris and Tableau is to extend the declarative principle of SQL by visualisation, inspired by the fact that SQL already offers some abilities to structure, sort and group data. In order to find a compromise between ease of use and flexibility, Tableau concentrates on tabular data and uses the familiarity of users to spreadsheets. Queries are defined in VizQL (cf. Sect. 4.2.1), a visual query language to visually construct queries to relational databases and OLAP servers, generating SQL. Via drag-and-drop, columns can be selected as input to the visualisation process. By adding fields from the database to either columns or rows of a metaphorical table representation, the user can influence how these fields are nested. This allows, for example, for the description of *small multiple*[1] views of data.

Unlike Tableau, our focus is not on tabular data, but on heterogeneous ontological data.

SPSS Viz Designer

In Sect. 4.2.1 we discuss Wilkinson's formal *Grammar of Graphics*. A commercial Java system based on Wilkinson's grammar is *nViZn*. It is described as »*based on the mathematical definition of the graph of a function*« [WRRN01]. As opposed to Tableau or Polaris, it is not bound to relational data and can generate new graphics more flexibly. Since it was developed for the web, heterogeneous and distributed data sources are supported and network data can be processed. Unlike Tableau or Polaris, the available types of graphics include node-link diagrams. Also interaction can be specified: Filtering, sorting, linking and brushing between multiple views, zooming, and coordinate transformations (lenses) can be employed. While viewing data, the views can be interactively manipulated via widgets such as sliders.

Another software based on Wilkinson's algebra is *SPSS Viz Designer*. It was added as a visualisation component to *SPSS*, a statistic analysis and visualisation tool by IBM, which is

[1] A nested graphic representation showing multiple small, similar graphics next to each other.

mainly focused on quantitative data. Viz Designer uses a tabular representation of data where the rows are the cases and the columns are the variables. In contrast to Microsoft Excel, SPSS Viz Designer allows the assignment of additional information to variables, including the scale of measurement. In cases where the settings that can be specified via the user interface are not sufficient, the user can write more complex configurations using one of the languages ViZml and GPL (cf. Sect. 4.2.1).

Although not only tabular data is supported, neither nViZn nor Viz Designer considers the specifics of ontological data.

TopBraid Composer + UISPIN

TopBraid Composer is a Semantic Web development environment and part of the TopBraid technology stack[2]. It supports an ontology-aware node-link view of RDF-graphs. Beyond this, RDF-data can be rendered to document shape (HTML and SVG) using UISPIN[3], which we introduce in the languages part (Sect. 4.3.1). UISPIN builds on SPIN, which we introduced in Sect. 2.2.7.

While TopBraid Composer + UISPIN covers many of our requirements and brings reusable components, it has only basic visualisation capabilities, particularly with respect to the composability of graphics.

Cytoscape

Cytoscape [SMO+03] is an open-source plug-in environment for biological network visualisation and analysis actively used in bioinformatics. Layouting and data filtering are core functionalities the framework offers. Many analysis plug-ins have been developed for it, including RDFScape, a plug-in for RDF visualisation (described below).

The most interesting part of the framework with respect to our work is the *Vizmapper* (cf. Fig. 4.1). The Vizmapper allows for the advanced mapping of network attributes to visual attributes via *visual mappers*. These mappers come in the variants: *continuous mappers*, *discrete mappers* and *continuous to discrete mappers*. It is also possible to define some visual attributes to depend on the values of other attributes, e. g., *width = height*, *node colour = text colour*, *arrow colour = line colour*. Legends can be automatically generated. Further features are filters for data customisation and some basic editing functionality (creating, deleting nodes).

RDFScape

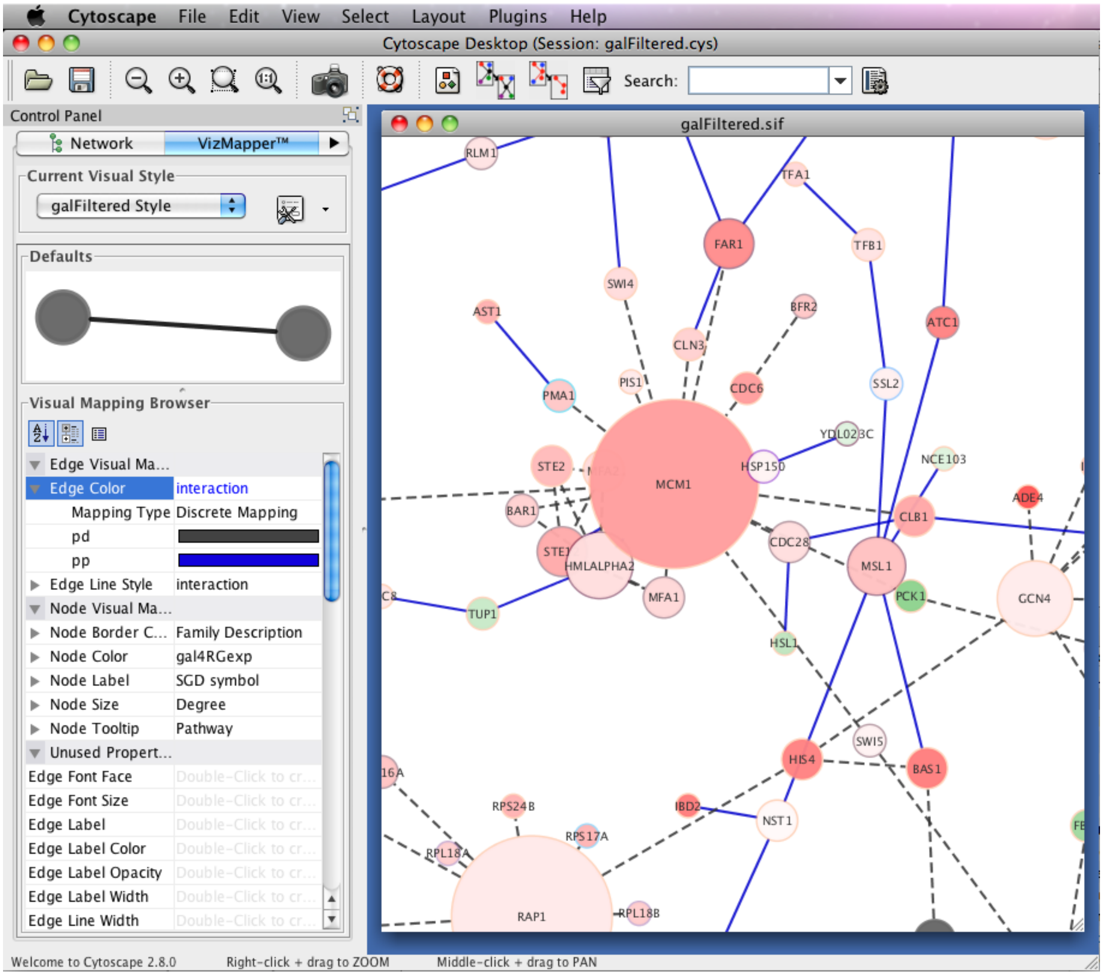
RDFScape [Spl08] is a plug-in for Cytoscape that enables the display of RDF graphs. Since RDFScape is intended for use in the field of biotechnology, it concentrates, like Cytoscape, on node-link representations that are a seemingly obvious choice for interaction networks and similar data.

Mapping features are the following: In addition to the node-link representation, datatype properties can be displayed like UML-attributes (a table contained in a node), and properties such as rdfs:label or URIs can be displayed as labels of nodes. Furthermore, different namespaces can be mapped to different colours. All mapping settings can conveniently be specified using the Cytoscape Vismappers. Other features include querying ontologies with SPARQL and RDQL, incrementally extending (»navigating«) the graph visualisation based on the ontology, and using the graph visualisation to define visual queries. The system supports (customisable) reasoning and the inferred knowledge can be filtered and visualised as well.

RDFScape does not offer guidance functionality for the visual mapping. Also, the mappings themselves cannot be stored as RDF for inter-tool exchange.

[2] TopBraid. http://www.topquadrant.com/technology/topbraid-platform-overview/, accessed: 09.11.2015.

[3] UISPIN was renamed to SPARQL Web Pages (SWP).



Current Visual Style: galFiltered Style

Visual Mapping Browser

Edge Visual Ma...	
Edge Color	interaction
Mapping Type	Discrete Mapping
pd	■
pp	🟦
Edge Line Style	interaction
Node Visual Ma...	
Node Border C...	Family Description
Node Color	gal4RGexp
Node Label	SGD symbol
Node Size	Degree
Node Tooltip	Pathway
Unused Propert...	
Edge Font Face	Double-Click to cr...
Edge Font Size	Double-Click to cr...
Edge Label	Double-Click to cr...
Edge Label Color	Double-Click to cr...
Edge Label Opacity	Double-Click to cr...
Edge Label Width	Double-Click to cr...
Edge Line Width	Double-Click to cr...

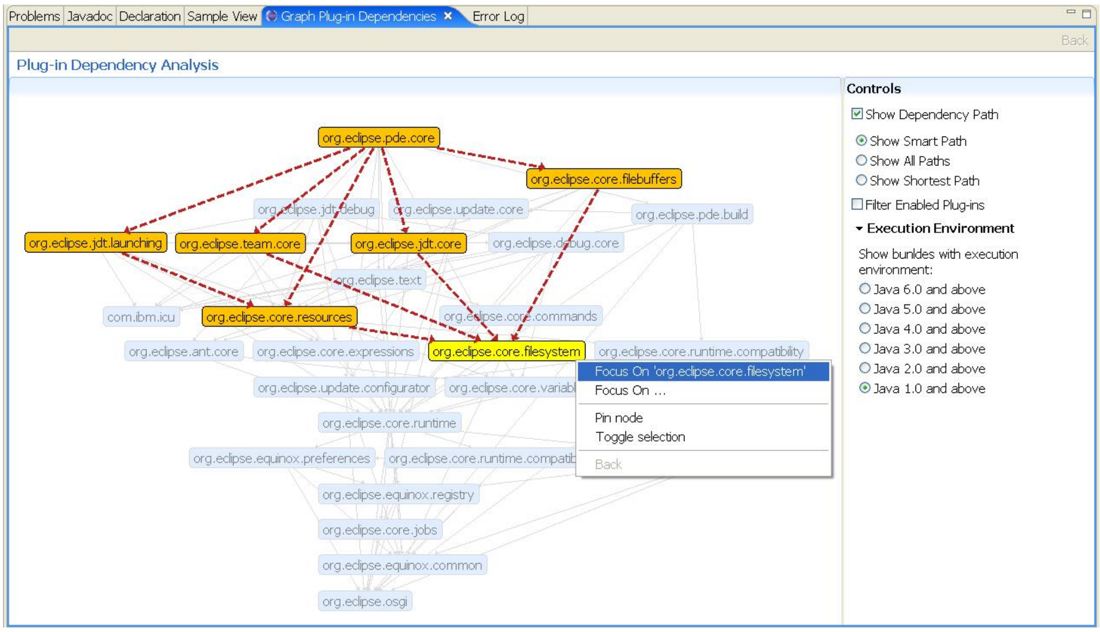
Welcome to Cytoscape 2.8.0 Right-click + drag to ZOOM Middle-click + drag to PAN

Figure 4.1: Cytoscape – Visual Mapper View.

Model-Driven Visualisation – Bull, 2008

In the Model-Driven Visualisation approach (MDV) [Bul08], Bull uses a model transformation language (ATL[4]) to build view (visualisation) models from domain models, both represented in Ecore (cf. Sect. 2.2.6). The view model is realised with the ZEST[5] toolkit the author initiated. ZEST generalises the principle of using the Model View Controller (MVC) pattern [Bur92] and *content providers* (interface IContentProvider), which has been used before in SWT/JFace widgets such as trees and lists, and transfers it to other graphics such as node-link representations (cf. Fig. 4.2). Both the view model and the controllers (the content providers) for the MVC pattern are generated by MDV.

However, MDV alone does not provide an approach to visually support the configuration of visual mappings and can only be used for the visualisation of ontologies with the extensions described in the next subsection.


Problems
Javadoc
Declaration
Sample View
Graph Plug-in Dependencies
Error Log
Back
Plug-in Dependency Analysis
org.eclipse.pde.core
org.eclipse.core.filebuffers
org.eclipse.jdt.debug
org.eclipse.update.core
org.eclipse.pde.build
org.eclipse.jdt.launching
org.eclipse.team.core
org.eclipse.jdt.core
org.eclipse.debug.core
org.eclipse.text
com.ibm.icu
org.eclipse.core.resources
org.eclipse.core.commands
org.eclipse.ant.core
org.eclipse.core.expressions
org.eclipse.core.filesystem
org.eclipse.core.runtime.compatibility
org.eclipse.update.configurator
org.eclipse.core.variabl
org.eclipse.core.runtime
org.eclipse.equinox.preferences
org.eclipse.core.runtime.compatib
org.eclipse.equinox.registry
org.eclipse.core.jobs
org.eclipse.equinox.common
org.eclipse.osgi
Focus On 'org.eclipse.core.filesystem'
Focus On ...
Pin node
Toggle selection
Back

Controls

☑ Show Dependency Path

● Show Smart Path
○ Show All Paths
○ Show Shortest Path

□ Filter Enabled Plug-ins

Execution Environment

Show bunldes with execution environment:

○ Java 6.0 and above
○ Java 5.0 and above
○ Java 4.0 and above
○ Java 3.0 and above
○ Java 2.0 and above
● Java 1.0 and above

Figure 4.2: ZEST – Static Graph Viewer showing Eclipse plug-in dependency graph. Taken from http://www.eclipse.org/gef/zest/.

CogZ + MDV

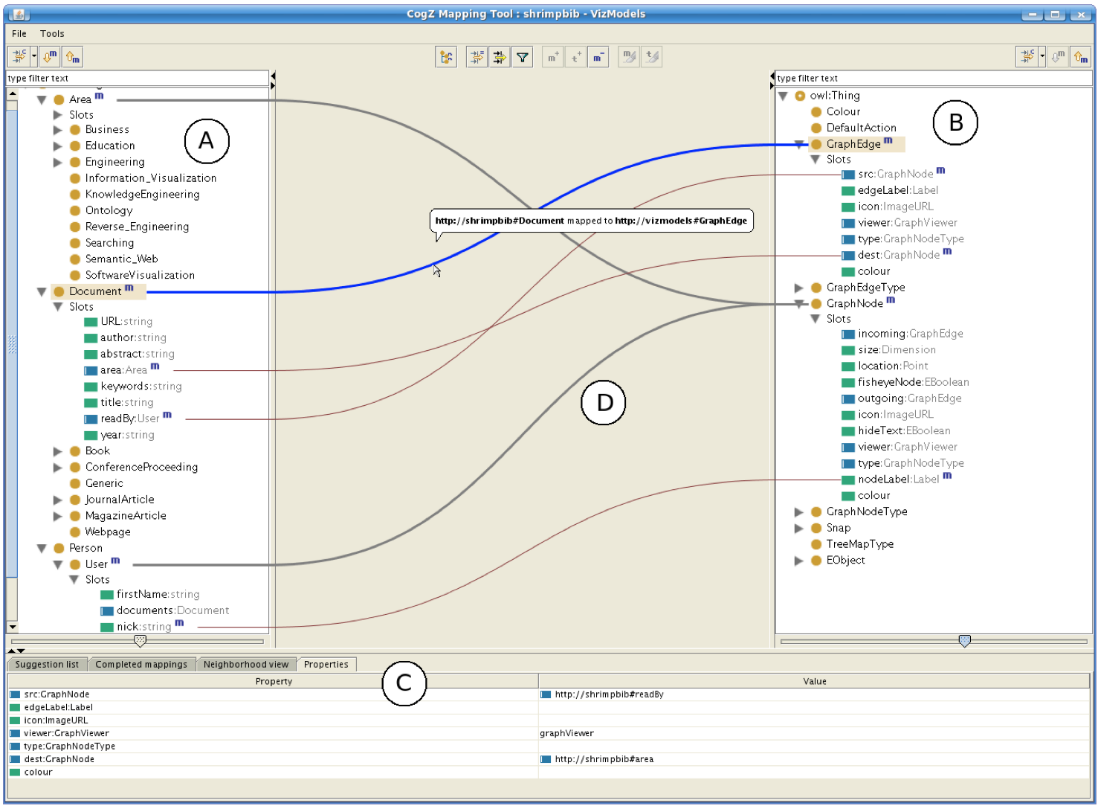
An approach for using MDV for ontologies as well was developed by Falconer and Bull et al. at the interdisciplinary CHISEL group (University of Victoria) [FBGS09], extending the existing, Protégé-based ontology mapping tool CogZ [FS07]. Thereby, they gained a visual mapping interface for defining a visual mapping of ontology terms to terms of a visual model, which is also defined as an ontology and which describes one of the models that the MDV approach can handle (cf. Fig. 4.4). It is then possible to generate a visualisation of the domain instance data.

CogZ stores mappings explicitly, but only rather simple value mappings are possible. Although instances could be mapped directly (e. g., »water« mapped to »blue«, »location« mapped to »square« . . .), problems arise, when mapping to continuous ranges. These mappings cannot be

[4] Atlas Transformation Language. http://www.eclipse.org/atl/, accessed: 11.11.2015.

[5] ZEST, Eclipse Visualization Toolkit. http://www.eclipse.org/gef/zest/, accessed: 10.10.2015.

easily defined with the CogZ plug-in. Furthermore, no guidance is offered on which visualisation options are most appropriate.



http://shrimpbib#Document mapped to http://vizmodels#GraphEdge

Property	Value
src:GraphNode	http://shrimpbib#readBy
edgeLabel:Label	
icon:ImageURL	
viewer:GraphViewer	graphViewer
type:GraphNodeType	
dest:GraphNode	http://shrimpbib#area
colour	

Figure 4.3: CogZ+MDV – Mapping View. Taken from [FBGS09].

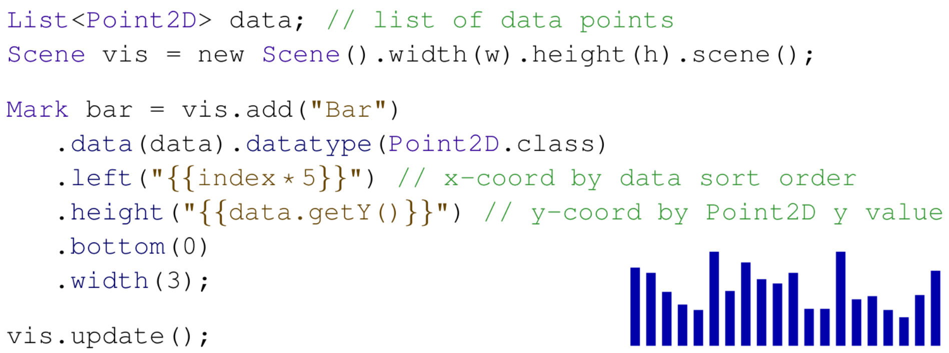
Generative Software Visualization – Müller et al., 2011

Müller et al. suggest an approach to generative software visualization [MKSE11] that is related to the work of Bull. Both are based on model-driven technologies located in the EMF technical space and introduce platform-independent intermediate models. However, for our detailed tabular comparison, we chose the approach from Bull. We consider it closer to ours, because it is domain-independent and not specifically about visualising software. Having said this, the approach of Müller et al. has also similarities with the OGVIC approach: Unlike Bull, they use a DSL to avoid writing transformations in multi-purpose (transformation) languages. Similarly, we introduce the (domain specific) language RVL to simplify the mapping definition (Chapter 7). Besides that we do not focus on the domain of software visualisation and exclude the aspect of 3D visualisation in this work, the OGVIC approach differs in the way that mappings are defined. Although Müller et al. employ clear mapping rules (e. g., represent attributes as nodes shaped as a blue cone) they do not distinguish the mappings of relations (in the domain data) from the mapping of values as we do. Neither the approach of Bull, nor the one of Müller et al. store general visualisation knowledge and rules externally in standard formalism (to allow for reuse). Instead, this configuration knowledge is part of the generators.

Protovis

Protovis, developed at the Stanford Visualization Group, is a JavaScript visualisation toolkit with a declarative visualisation language also being referred to as Protovis [BH09]. Since implementations in JavaScript (JS) and Java are offered, we do not only discuss it as a *language* but included it in the list of visualisation approaches (cf. Sect. 4.2.1 for details on Protovis as a language). Fig. 4.6 gives an example of a declarative visualisation definition of a bar chart in Protovis (Java version) and also shows the corresponding rendered SVG+HTML result on the right.

The development of Protovis was stopped in the meantime in favour of the follow-up project D3[6] [BOH11]. While the declarative approach and the support of interactivity and incremental updates are close to our needs, neither Protovis nor D3 support ontological data.



```
List<Point2D> data; // list of data points
Scene vis = new Scene().width(w).height(h).scene();

Mark bar = vis.add("Bar")
    .data(data).datatype(Point2D.class)
    .left("{{index*5}}") // x-coord by data sort order
    .height("{{data.getY()}}") // y-coord by Point2D y value
    .bottom(0)
    .width(3);

vis.update();
```

Figure 4.4: Protovis (Java version) – Bar chart example. Taken from [HB10].

SemViz

The approach of Gilson et al., called *SemViz* [GSGC08], aims at automation for ontology visualisation. The authors tried to achieve this by combining an existing mapping algorithm with probabilistic reasoning techniques.

Within the mapping process, the system uses three ontologies: The first is a Domain Ontology (DO) to describe the domain data semantically. Second, a Visual Representation Ontology (VRO) captures a description of a visual representation (2D Graphs, TreeMaps, and Parallel Coordinates are shown in the examples). And third, a Semantic Bridging Ontology (SBO) stores expert knowledge on the appropriateness of mapping a domain concept to some visual attribute.

In a first step, for each domain, a DO has to be created by a system (not a domain) expert and is then mapped automatically to tabulated data that has been extracted from semi-structured webpages. Since we assume to have structured data as a starting point for visualisation, we do not further discuss this step. In a second step, a »controlled set of relationships and attributes« has to be defined and used in the DO. These are very general and include statements on the scale type (qualitative, quantitative, informational), information concerning primary keys, priority and complementation. The VROs describe the properties of the visual representation, e. g., *X*, *Y*, *Width*, *Height*, *Text Label*, *Shape Colour*. Again, relations between these properties are given such as priority, complementation and statements on the (accepted) scale type of data. There is an attribute isQuantitative that is used to describe domain concepts in the DO and a »semantically equivalent« attribute that is used to describe »visual artefacts« (visual means) of the VRO. For example, the concept *Weeks in Chart*[7] has the value *0.75* for isQuantitative in the DO and, in the VRO, *Y* is assigned a value of *0.99* for isQuantitative. Similarly, complements is used in the DO and in the VRO. For example, the concept *Weeks in Chart* only complements

[6] D3.js (Data Driven Documents). http://d3js.org/, accessed: 02.07.2015.

[7] We are referring here to the example domain ontology used by the authors, which is about music charts, songs and artists.

Last Week Chart Position with a value of *0.6* (DO), while *Y* complements *X* with a value of *0.9*, making *Last Week Chart Position* not a perfect match for *Y*. Finally, based on the »semantically equivalent« relations, a matching of DO and VRO concepts is then performed, leading to a set of recommended permutations. To reduce the number of permutations, additional expert knowledge can be represented in the SBO by weighting some of the so called »semantic bridges« between DO and VRO concepts with higher appropriateness values than others (cf. Fig. 4.5).

Although Gilson et al. even aim at »automatic generation of visualizations« [GSGC08], they as well offer automation only to some extent. For example, a specific domain ontology has to be created manually for every new domain. Also, expert knowledge has to be formalised manually in a SBO for each new combination of a new DO with a VRO. As opposed to this, we aim at semi-automation for arbitrary domains, leaving the actual mapping to the user, but trying to support her as much as possible inspecting characteristics of the existing domain ontologies. Another difference between our approach and SemViz is that we do not (directly) match domain relations and visual means by scale types, but take into account what is the expressiveness and effectiveness of a visual means for a given scale type.

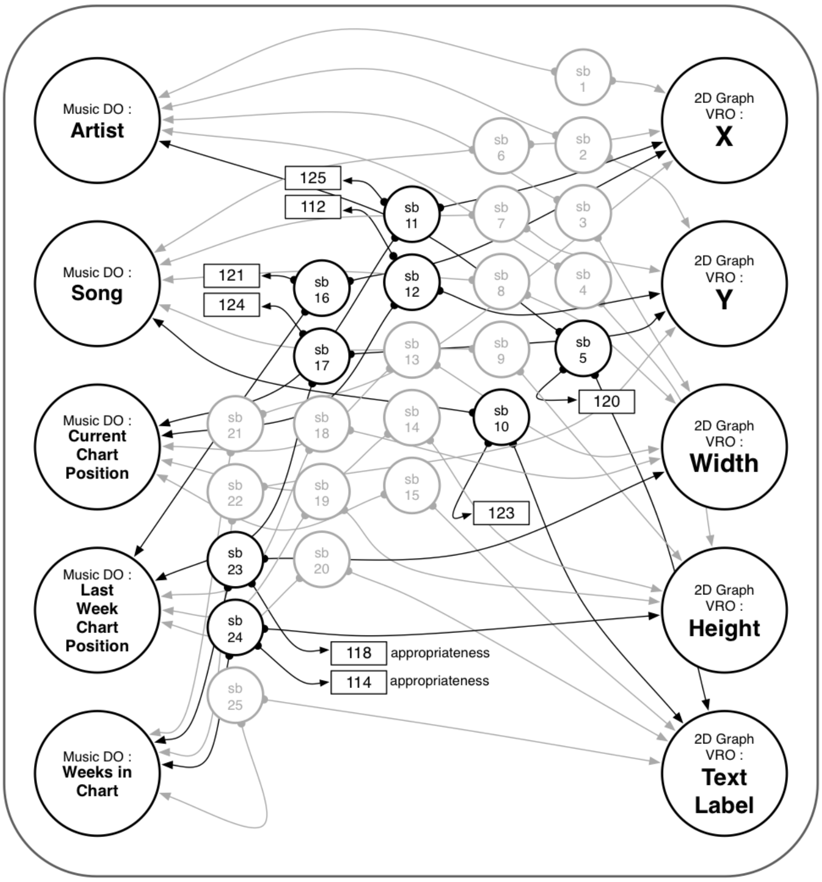


Figure 4.5: SemViz – Semantic Bridging Ontology. Taken from [GSGC08].

Less + LeTl

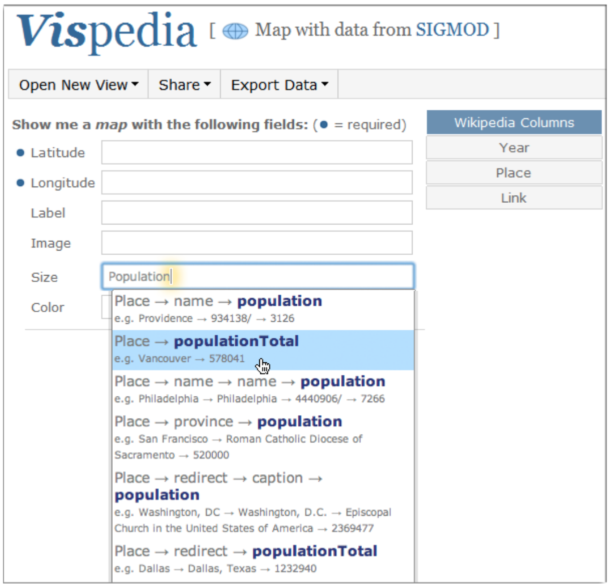
Less [ADD10] is a collaborative, template-based system, mainly for presenting RDF from SPARQL-query results. The authors of LESS want to offer an end-to-end-approach and focus on community aspects. They, therefore, aim at a simple user-interface-based creation of templates and offer features such as dereferencing URIs, or caching to avoid dead links. Templates are described with the language *LeTl*, which we discuss separately in Sect. 4.3.1.

As opposed to the OGVIC approach, Less does not aim at visualisation but on presentation of data, i. e., visual mappings cannot explicitly be specified.

Vispedia

Cammarano et al. [CDC+07] describe a system that automatically matches RDF properties with attributes that visualisations offer. They see this problem from a schema matching perspective and try to find data variables that can be associated with the visual attributes of visualisation templates considering both structural matching and type matching. As templates, ready-to-use widgets such as the SIMILE timeline component are used.

Vispedia [CTW+09, CWT+08] builds on the work from Cammarano et al., presented above. Attributes of a visualisation are annotated with XML data types. Additionally, for each visual attribute, the user can provide a keyword. Based on this information, schema matching is used to match visualisation attributes with table columns of Wikipedia data tables and associated RDF data. The system provides multiple matching solutions (also following links in the RDF graph), from which the user can interactively choose the most appropriate one (cf. Fig. 4.6). Compared to our bottom-up (cf. Sect. 2.1.5) approach, which supports the synthesis of graphics, Vispedia could be referred to as a top-down approach, since only ready-to-use templates, i. e., complete graphic representations, may be used. The drawback of this approach is the lack of flexibility and composability.



Show me a *map* with the following fields: (● = required)

● Latitude

● Longitude

Label

Image

Size: Population

Color

Place → name → **population**
e.g. Providence → 934138/ → 3126

Place → **populationTotal**
e.g. Vancouver → 578041

Place → name → name → **population**
e.g. Philadelphia → Philadelphia → 4440906/ → 7266

Place → province → **population**
e.g. San Francisco → Roman Catholic Diocese of Sacramento → 520000

Place → redirect → caption → **population**
e.g. Washington, DC → Washington, D.C. → Episcopal Church in the United States of America → 2369477

Place → redirect → **populationTotal**
e.g. Dallas → Dallas, Texas → 1232940

Figure 4.6: Vispedia – Interactive mapping interface. Taken from [CWT+08].

4.1.2 Detailed Comparison by Criteria

We now compare ten approaches introduced in the last section by several criteria we chose in order to understand what are the most important differences between these approaches. Each criterion and its possible values will be explained in tabular form. Besides pointing to the most interesting characteristics of each approach, we present overview tables for groups of criteria.

All comparison tables share the following legend:

x	applies	(x)	applies with restrictions
(–)	not applies with some exception	–	not applies

Supported data models

As a first characterisation of processable data, we distinguish, whether **databases** *(C-1)* or **graphs** *(C-2)* are supported. While almost all approaches can represent graph data somehow (possibly as a three-column-table), they often do not support graph file formats. Especially the two complex visualisation design systems in our test set (Tableau and SPSS Viz Designer) are specialised on tabular, relatively homogeneous, data that comes in rows and columns from spreadsheets and databases. Although node-link-diagrams and matrices can be created with both systems, graphs are not in the focus of these tools. For example, in SPSS Viz Designer there is no support for handling graphs in the UI, and in Tableau, coordinates have to be given explicitly and operations on the graph such as joining nodes are not provided.

Awareness of structure and semantics

By using the term »awareness« instead of speaking of the »ability to represent some kind of data«, we want to stress that for this criterion, the ability of referencing constructs of the schema and creating specific visualisations matters. For example, a generic visualisation of an (RDF) graph, we do not consider ontology-aware.

Three levels of awareness are examined here: First: Can **constructs of a schema or types** be referenced? *(C-4)* Second: Can **ontology constructs** be referenced (in some ontology language)? *(C-5)* Third: If ontology constructs can be referenced, can also relations between classes and properties – **the T-Box** – be referenced? *(C-6)*

Except for Cytoscape, RDFScape and Protovis, which only have an understanding of typed graphs, all other approaches can reference constructs from some kind of schema. This may be a database schema or an XSD. Some of the approaches, e. g., MDV + Cogz or SemViz also allow for referencing constructs from ontologies, but not on the T-Box level (for example, characteristics of properties cannot be referenced). Only two of the approaches, TopBraid C. with UISPIN and Less + LeTl can also reference T-Box terms.

All the approaches in our test set that work with ontological data also **support RDF** or RDF-based languages such as RDFS or OWL *(C-3)*. For a definition of ontology and the classification of data we use here, refer to Sect. 2.2.

Domain

Since we did not restrict ourselves to a certain domain for visualisation, also the criterion **domain agnostic** *(C-7)* is important for the review of related work. None of the approaches is totally specific to a certain domain – however, we need to confine that statement for SemViz and Cytoscape. Cytoscape, having a biotechnological background, is focused on biological graphs such as gene–protein interaction networks. In SemViz a Domain Ontology (DO) needs to be created for every new domain, so that we cannot simply switch to a new domain. Besides this, some approaches have domain specific enhancements, e. g., Protovis recognises data from the geological domain. We return to this issue when comparing data analysis features in Sect. 4.1.2.

Supported data models

C-1 Relational databases supported The system can display/visualise data with a tabular data model.

C-2 Graphs supported The system can display/visualise data that forms a network.

C-3 RDF supported RDF or the languages built on top of it (RDFS, OWL) are supported.

Awareness of structure and semantics

C-4 Schema-/type-aware The system can reference the data's schema within display definitions. For graphs, edges are distinguished by type at least.

C-5 Ontology-aware (A/T-Box) The system can reference ontology terms (in any formalism) within display definitions. The visualisation is specific to ontologies. Representing RDF on a graph level is not sufficient.

C-6 Terminological ontology relations (T-Box) supported Specific representation of statements on the relation between classes and properties – T-Box (terminological component). Here it is not sufficient that these relations are somehow displayed, but they should be displayed differently from A-Box information.

Domain

C-7 Domain agnostic The system can display data from every domain. It has at least some support that is independent from the domain. Still there might be additional support for some domains.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Rel. DB supported	x	x	-	x	-	x	-	-	-	x
Graphs supported	-	(x)	x	x	x	x	x	x	x	x
– RDF supported	-	-	x	-	x	x	-	x	x	x
Schema-/type-aware	x	x	x	-	x	x	-	-	x	x
Ontology-aware	-	-	x	-	x	x	-	-	x	(-)
– T-Box support	-	-	(-)	-	-	-	-	-	(-)	-
Domain agnostic	x	x	x	x	x	(x)	(x)	x	x	x

Table 4.1: Comparison of related visualisation systems and approaches – Data.

Visual Mapping

We need to distinguish approaches turning data into a formatted document shape from those actually mapping information directly to the available visual means, which we defined as »visual mapping« in Sect. 2.1.6 *(C-8)*. Most approaches we selected for the comparison do at least some kind of visual mapping. Still we added some approaches that do not (or only in a limited way), but provide many other features of importance with respect to our requirements. For example, RDFScape is not able to implicitly perform a dynamic, value depending visual mapping from a set of data values to a set of visual values. This is only possible explicitly for each single value.

Interactivity

Interactivity is an essential part of the visualisation process if large amounts of data need to be visualised [YaKSJ07, Kei02, Few09]. Not only to enable overview-and-detail techniques as already stressed by Shneiderman's *Visual Information Seeking Mantra* [Shn96], but also for other techniques, such as *linking and brushing* [Kei02], which can be used to visually select items presented in multiple connected views.

> » The effectiveness of information visualisation hinges on two things: its ability to clearly and accurately represent information and our ability to interact with it to figure out what the information means.
>
> *Stephen Few [Few09]* «

Interactivity comprises the ability to have control/behaviour integrated with the visualisation *(C-9)*. Simple interactions are behaviours such as *mouseover* that can extend the palette of visual means with conditional display options. More powerful controls allow for turning the visualisation into a self-adaptive piece of software, since they may be used to interactively (re)configure the visualisation at each configurable transformation step of the visualisation reference model (cf. the next paragraph on *configurability*). In contrast to other ways of configuring a visualisation, the configuration with an interactive UI, puts requirements on the performance on the system, because very short response times between a users action (change configuration) and the system reaction (perform changes and show them to the user) are required [Sta06]. Thereby, the performance issue becomes a qualitative requirement and puts constraints on the chosen software architecture.

While all approaches support the definition of some kind of control/behaviour for the graphic that is produced, this is often limited to rather basic interactions, such as showing pop-ups on *mouseover* or *click* GUI events. In Sect. 4.2.2, we look at how interaction can be defined in visualisation languages.

Configurability

Our criteria for configurability are oriented to the classification of »customisations« of visualisations introduced in [Bul08]. Bull states various customisation aspects and methods and a classification of visualisations by customisation properties.

Visual Mapping **C-8 Visual Mapping (not only presentation)** The approach allows the definition of dynamic and value-dependent mappings to visual means. As opposed to this, presentation or styling only allows for statically and explicitly assigning visual values to some concrete items.

Interactivity **C-9 Interactivity** The output is interactive as opposed to static output such as PDF.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Visual mapping (not only presentation)	x	x	(-)	x	x	x	x	(x)	-	(x)
Interactivity	x	x	(x)	x	x	x	x	x	(x)	(x)

Table 4.2: Comparison of related visualisation systems and approaches – Visualisation and interaction.

Bull distinguishes three *aspects of customisation*:

- Data customisation
- Presentation customisation
- Control/behaviour customisation

Further, he distinguishes the following three *methods of customisation*:

- User Interface (UI)
- Domain Specific Language (DSL)
- Source code (SC)

While configuration by means of a UI, especially by a GUI, requires the least skills from the user with respect to the specifics of the data format and programming – unfortunately, it also offers the least flexibility. On the other end of the spectrum, there is the configuration via changes to the source code, which is highly flexible, but cannot be expected to be done by domain-experts. Using DSLs is situated somewhere in between both extremes and can be an appropriate choice for some user groups, but still requires some training to be used efficiently. The chosen method of configuration is inherently connected to the criterion of interactivity: While small, precisely defined changes via a GUI or DSL may be used to interactively change a configuration, writing new source code does not lead to an interactive experience with short system response times. Similar to Bull, we set up three criteria of configurability – data configurability, presentation configurability and control/behaviour configurability:

Data configurability *(C-10)* comprises filtering data, renaming data items and sometimes also deriving new information to be presented. For information visualisation, filtering data is important in order to reduce large data sets to the most important aspects. Table 4.3 shows that most of the approaches allow for some customisation of data. However, frequently this cannot be achieved with an interactive UI, but has to be done via DSLs or by writing additional source code. An example for more advanced filtering is Cytoscape. It supports the construction of conjunctive queries and optionally negation. Also ranges can be selected for ordinal data. Still the interactive filtering experience is limited here, because settings have to be submitted before changes are performed.

Presentation configurability *(C-11)* is achieved by all approaches, mostly via a GUI. Still, there are other approaches such as Protovis that uses an embedded DSL (EDSL) to achieve higher flexibility. Also *Less* requires users to write template code, although the management and sharing of templates is supported with a GUI. In SemViz, no actual »configuration« is foreseen at all, but a change in the presentation can be achieved indirectly by varying probability settings.

Besides configuring visual mapping settings, some approaches also support view modifications. For example, Cytoscape allows for hiding elements and bypassing the mapping with constant values.

Control/behaviour configurability *(C-12)* is often supported only marginally by means of source code or DSL configuration. While simple behaviours such as mouseover effects can sometimes be defined explicitly, more complex interactions can only be achieved via scripting (e. g., Protovis, Less, UISPIN).

Target user group

Since we did not restrict our target users to visualisation experts or programmers, also the average user, who may be a domain-expert, should be able to use it. Therefore, it is interesting to consider the target user group of other visualisation approaches. We found that only three of the examined tools, Tableau, Viz Designer and Vispedia may be **used by domain experts** without visualisation or programming experience *(C-13)*. In all other cases, knowledge on visualisation, programming or confidence with semantic web languages is required.

Explicit Presentation Definition

We distinguish whether presentation information is stored internally or single views/mappings are explicit *(C-14)*, i. e., they are entities given a name, maybe even a URI. Explicit mappings are the prerequisite for many of the following characteristics, such as variability, composability and shareability. Almost all approaches we examined bring their own languages for explicitly defining presentation information. These languages will be the subject of the following two sections 4.2 and 4.3. However, there are a few exceptions to this: In SemViz, the final mapping is automatically calculated from probabilities and not defined explicitly. Cytoscape and RDFScape use internal properties-files for the purpose of storing presentation settings. Each mapping is a line in a property-file that cannot be externally referenced. Also Vispedia does not offer the possibility to name the single mappings that make up a visualisation.

Variability

We distinguish two dimensions of variability – platform variability and variability of the visual structure. For both, explicit presentation definition is a precondition, since we need to be able to refer to the display definitions in order to transform or replace them.

Platform Variability Platform variability *(C-15)* is achieved by describing the definition of presentation in a platform-independent way. Two ways are possible. The first way is a generative approach. Instead of directly constructing a visualisation on the final presentation platform (such as HTML, SVG, X3D or Java2D), a platform-independent model is created that can be further transformed into platform-specific code in a second step. The MDV + Cogz approach and SemViz take this way. Also Protovis (JavaScript version) is able to generate visualisation code for three platforms, HTML5 Canvas, SVG and Flash [BH09], from definitions given by means of the JavaScript-embedded DSL. Protovis (Java version) even has an event model that is decoupled from the runtime platform in order to support platform independence [HB10]. The

Configurability

C-10 Data Configurability (incl. filtering) Relevant information (for the users task) can be separated from the rest. Possibilities are: Filtering, data deduction, move/rename data (cf. [Bul08]).

C-11 Presentation Configurability (incl. visual mapping) The selection and configuration of visual means is offered. Additionally views may be adjustable (cf. [Bul08]).

C-12 Control/Behaviour Configurability Some behaviour of visualisation parts and interactive controls can be configured easily.

Target user group

C-13 Domain experts can use the system Users need to have only little background knowledge on visualisation or programming but can use their world knowledge to configure visualisations.

Configurability	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Data conf.	GUI	GUI	DSL	SC	-	SC*	GUI	GUI	DSL	GUI
Presentation conf.	GUI	GUI	GUI	EDSL	GUI	SC*	GUI	GUI	SC	GUI
Control/behaviour conf.	-	(DSL)	SC	EDSL	-	-	(-)	(-)	SC	-
Usable by domain experts	x	x	-	-	-	-	x	(-)	-	x

(E)DSL = Configuration method: (Embedded) Domain Specific Language
GUI = Configuration method: Graphical User Interface
SC = Configuration method: Source Code
*) by modifying the code for extracting data from webpages or the stored probability settings

Table 4.3: Comparison of related visualisation systems and approaches – Configurability.

second way is the interpretative approach, in which multiple variants of a system for different platforms are built, which interpret a common, platform-independent, declarative presentation definition. The visualisation design systems Tableau and Viz Designer take this approach in order to allow for desktop and web renderings.

However, many approaches can only define presentation information for a single platform. Especially the graph visualisation design systems Cytoscape and RDFScape do not create platform-independent presentation definitions, nor do the web-based ones (Less and Vispedia). The lack of platform-independent models in some systems does not mean that this could not be beneficial: For example, developers of Cytoscape discussed a platform-independent model they call »ViewModel« for a new version of Cytoscape in order to achieve independence from the rendering engine, however, this has not been implemented by now[8].

Visual Structure Variability As an example of visual structure variability consider the following case: »We want to vary our visualisation by selecting a tabular representation instead of a node-link representation.« This change of the graphic representation implies that one visual structure (in this case the dominant one) has to be replaced from *linking* to multiple *line-ups* and *separations* making up a table. Despite that change, we would like to keep other mapping definitions, such as mappings to colour or size or even mappings to other (minor) visual structures.

A prerequisite of this criterion is the general availability of **multiple explicitly defined visual structures** to choose from *(C-16)*. TopBraid C. + UISPIN and Less+LeTl do not offer the selection of alternative visual structures, but use a box model that can be used well to render nested HTML DIV-elements. Similarly, Cytoscape and RDFScape are bound to the node-link paradigm that can easily represent graphs. All other approaches support multiple visual structures, usually by offering alternative graphic types such as »Matrix«, »Node-link«, »Tabular«, »Tree Map« or »Bar Chart«. If an approach supports the creation of multiple visual structures only by means of a natural purpose language, we do not consider the criterion fulfiled, because one cannot switch between multiple explicitly defined structures.

Similar to platform variability, we now ask if we can **keep other mapping definitions when we switch to different visual structures**[9] *(C-17)*. Of course, this can only work when there are some settings that are valuable for multiple visual structures. Vispedia and SemViz support multiple visual relations, but these cannot be varied independently from the main visual structure, since a completely different template or ontology representing a different type of graphic has to be selected. All other approaches allow for at least some definitions to be reused when changing the visual paradigm. For example, Viz Designer allows for shifting from a bar to a dot diagram without loosing the configuration of visual attributes (such as colour settings).

Composability of the Defined Views/Mappings

The composition of visual mappings and views needs to be distinguished from the composition of graphics from graphic objects [vE02]; see also Sect. 8.2. For the field of visualisation, the composition of graphics has been described by the algebras of Mackinlay [Mac86a] and Wilkinson [Wil05]. Also Card [CM97] discusses various classes of »*composing a compound Visual Structure out of several simple Visual Structures*«. However, here we refer specifically to the composability of visual mappings and views. For the purpose of this analysis, we define composability as *(C-18)*. »Explicitness« of the views or mappings *(C-14)* is a prerequisite for this criterion. Furthermore, it also subsumes »reusability« of the components. Our definition of composability does not

[8] Cytoscape Discussion Forum. http://wiki.cytoscape.org/Cytoscape_3/CoreDevelopment/Architecture, accessed: 21.02.2012.

[9] This includes being independent from the dominant *main* visual structure, which has also been called visual paradigm [PBKL06b].

Explicit presentation definition	**C-14**	**Explicit view/mapping definition** By explicitly specifying a view/mapping it can more easily be reused, shared, found, modified, analysed and discussed by others; cf. [Bul08].
Platform variability	**C-15**	**Platform variability** Code for various platforms can be generated from a common description or the mapping/view definition is interpretable on various platforms.
Visual Structure Variability	**C-16**	**Multiple Visual Structures/Paradigms** Multiple visual structures/-paradigms or types of graphics can be chosen by the user.
	C-17	**Visual structure variability** Visual structures can be changed independently from other parts of the view/mapping definition.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemVIz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Explicit view/mapping definition	x	x	x	x	x	-	-	-	x	-
Platform variability	x	x	(x)	x	x	x	-	-	-	-
Multiple visual structures/paradigms	x	x	-	x	x	x	-	-	-	x
— Visual structure variability	x	-	-	x	(x)	-	-	-	-	-

Table 4.4: Comparison of related visualisation systems and approaches – Explicit presentation definition & variability.

Composition	**C-18**	**Composability of the defined views/mappings** Explicitly defined views/mappings can be combined by third parties to build more complex ones.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemVIz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Composability of the defined views/mappings	(x)	-	x	(x)	-	-	-	-	x	-

Table 4.5: Comparison of related visualisation systems and approaches – Composition.

directly relate to that of software components as given by Szyperski [Szy97], but we too require that composition by third parties is possible.

Since Protovis and Viz Designer allow for storing mappings only in variables, we list these two approaches as supporting »restricted composability«. Only two approaches among the related work fully support composability according to our definition: TopBraid C. + UISPIN allows for defining templates that can be used to instantiate more complex templates. Also Less+LeTl supports calling (named) templates within other templates. Our less restrictive definition of composability does not require the explicit definition of a composition system including a component model and a composition language, as propagated in [Aß03b], neither is this provided by any of the approaches.

While the criterion of composability is important for comparing visualisation approaches, we discuss composability again in the languages sections 4.2 and 4.3, since it is foremost a language criterion.

Degree of Automation

Visualisation systems can be characterised by their degree of automation. We take up this criterion in order to compare our approach that we see as semi-automatic, to existing work.

We speak of **manual visualisation** *(C-19)* if no guidance by the system is given to the user and no default settings are performed. Arbitrary visualisations, including ineffective and inappropriate ones, can be created.

Following Wills [WW10] we define **automatic visualisation** as full automation, where no interaction of the visualisation author is required anymore, once the data to be visualised has been passed to the system *(C-21)*. An example for an automatic visualisation system is the »Show me!« [MHS07] component of the Tableau visualisation design system. »Show me!« analyses the data to be visualised and offers only possible graphical representation types to the user. For example, when geographical data is recognised by the system, a map is offered as a potential choice. Otherwise, this option is greyed out. SemViz is classified as an automatic visualisation approach by the authors, according to their schema. However, in order to visualise data from a new domain, preparations have to be taken such as creating a new domain and semantic bridging ontology and the user has to select from a best-of gallery in a last step. That means, although the visualisation process runs automatically, extra manual preparations have to be done in advance. Wilkinson lists more examples of automatic visualisation tools: SAS, SPSS, Stata and SYSTAT, which are not discussed here in detail, since automation is not in our focus [Wil05].

Degree of automation	**C-19** **Manual** The system allows to manually create a visualisation from data. It does not constraint the users choice nor does it recommend something. Errors may happen and also ineffective and inappropriate visualisations can be created.
	C-20 **Semi-Automatic** The system partially acts automatically. The user still has some configuration to do, but some settings are put to defaults since some choices have been done automatically. The user is supported with the mapping step in the visualisation process. This support can be the suggestion of defaults, the issuing of warnings or the display of a help menu and also (but not necessarily) guidance.
	C-21 **Automatic** The user simply inputs data and receives a visualisation without configuring the system.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Manual	-	-	x	-	x	-	x	x	x	-
Semi-Automatic	x	x	-	(x)	-	x	x	-	-	x
Automatic	-	x*	-	-	-	-	-	-	-	-

*) The »Show Me!«-button aims at full automation.

Table 4.6: Comparison of related visualisation systems and approaches – Level of automation.

All other systems, where some part of the visual mapping has to be done manually by the visualisation author, we refer to as **semi-automatic** *(C-20)*. Here, default values may be set by the system but it may still ask for manual intervention.

Wilkinson states that »*combining contextual meta-data with the graphics grammar can enable us to generate graphics automatically from user queries*« [Wil05, p. 51]. Hence, for (semi-)automation to work, the system requires knowledge on both the data to process and the visualisations to create. We come back to these two issues in the following two subsections.

Consideration of Data Semantics

The first source of information that is needed to enable (semi-)automation is the data itself. Therefore, we compare how existing approaches extract, analyse and exploit qualitative and quantitative characteristics of the data, the data's schema and the available metadata. We distinguish, whether systems **consider the semantics of the data** at all *(C-22)* and if so, whether they gain this information **from inspecting an underlying ontology** *(C-23)*. This criterion is closely related to what other authors called »information assisted visualisation« [GSGC08].

Tableau and Viz Designer analyse source data, but do this by inspecting quantities or by guessing the data type. For example, Tableau uses the number of records to choose the type of graphic. Also, some semantic properties of a field, such as geographical and time data types are recognised. SPSS Viz Designer extracts some date-strings as dates, guesses the semantics of zip codes from their values and recognises geographical coordinates.

Additionally, to the extraction of semantics, systems may also ask the user in case of doubt and allow her to improve the systems decisions. In Tableau, after default settings are made based on the fields data type, a GUI lets the user choose how to interpret a given database field:

either as a *measure* or as a *dimension.* As Tableau, also Viz Designer allows the assignment of additional information to variables, including the scale of measurement that may be either *nominal, ordinal, or scale*[10].

However, none of the approaches considers complex axioms of an ontology. SemViz and Vispedia use ontologies, but only try to match data types and do not exploit further metadata such as relation characteristics.

Usage of Visualisation Knowledge

As stated above, (semi-)automation requires the system to have expert knowledge on visualisation. We distinguish whether a system **uses visualisation knowledge** at all, stored by any means *(C-24)*, or whether the **visualisation knowledge is stored in an ontology** *(C-25)*. The (semi-)automatic visualisation design systems (e. g., Tableau) have their own internal representation of knowledge that they use to make reasonable defaults and automate visual mapping. Also Protovis and Cytoscape have some built-in visualisation knowledge. However, none of these tools allows for exchanging this knowledge with new facts in a standard formalism. SemViz is the only approach that stores visualisation knowledge in an ontology, but here we have the special case that this knowledge consists of probabilities on good matches between visual attributes and domain specific attributes. No generally valid visualisation knowledge is used, but expert knowledge has to be newly defined each time the domain ontology or the desired visualisation changes.

Usage of Guidance

We distinguish three use cases of guidance in the context of visualisation, aligned with the stages in the visualisation pipeline model that allow for human interaction (cf. Sect. 2.1.4). Our main interest is on »guidance for visual mapping«. Still, we also inspected all systems with respect to how »guidance for data filtering«, being part of the data transformation stage, and »guidance for view transformation« is achieved. For a general definition of guidance cf. Sect. 2.3.

Guidance for filtering *(C-26)* is supported by about half of the systems we examined, though often very basically. In Cytoscape, filters are adapted based on the range of the possible values. RDFScape allows for the visual and guided creation of filter queries by graph patterns. Also Vispedia helps the user to filter the data he is interested in. In Tableau, *Quickfilters* and range widgets are generated. However, no approach integrates advanced self-adaptive visual filtering techniques, such as faceted browsing, to guide the user and avoid empty result sets. Also the semantics of the data are not used to further guide the filtering process.

Guidance for visual mapping *(C-27)* can mean that incomplete mappings are avoided and only possible values are allowed when visual mappings are created. Furthermore, guidance for the visual mapping process may include discouraging inexpressive visual means or suggesting the most effective visual means (cf. Sect. 5.7). We found that only Tableau, Viz Designer and Vispedia guide the user when doing the visual mapping. Also Cytoscape offers some very limited guidance by means of suggested defaults. Table 4.8 also shows that whenever guidance for visual mapping is supported, this is achieved **by a graphical user interface** *(C-29)*. CogZ offers an interesting UI for visual mapping where mappings are created visually by drawing connector lines, however no guidance is given here.

One example for **Guidance for view transformations** *(C-28)* can be found in SPSS Viz Designer, which offers guidance for some very basic view transformations, such as the activation of the 3D exploration mode for three-dimensional graphics. However, none of the other approaches guides view transformations.

In summary we can say, while guidance is offered quite often for data filtering purposes, the guidance of the visual mapping process is a feature that can be found in the established

[10] *Scale* here comprises ratio and interval scale

Consideration of the data semantics

C-22 **Considers semantics for visual mapping** When doing the visual mapping, semantics of the data are taken into consideration either by the system automatically, or they are turned into suggestions and restrictions for the user.

C-23 **Considers semantics of an ontology for visual mapping** Complex axioms in the data can be considered for visualising instance data in a more appropriate way. Not only the meaning of fields is considered somehow, but an ontology is inspected by the system. Guidance or automation is not a requirement to fulfil this feature.

Usage of visualisation knowledge

C-24 **Uses visualisation knowledge** The system uses formalised visualisation knowledge, such as colour palettes, rankings of visual means, facts on relations of visual means.

C-25 **Uses visualisation knowledge from ontology** The system uses visualisation knowledge that has been formalised as an ontology

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Considers semantics for visual mapping	x	x	n/a	-	-	x	-	(-)	n/a	(x)
– . . . of an ontology	-	-	n/a	-	-	(-)	-	-	n/a	(-)
Uses visualisation knowledge	x	x	n/a	x	-	(x)	(-)	(x)	n/a	-
– . . . from an ontology	-	-	n/a	-	-	x	-	-	n/a	-

n/a = Not applicable. These criteria are only applicable to approaches that support at least semi-automatic visual mapping.

Table 4.7: Comparison of related visualisation systems and approaches – Consideration of data semantics and usage of visualisation knowledge.

Usage of guidance	**C-26**	**Guidance for Data Filtering** The filtering of data is guided.
	C-27	**Guidance for Visual Mapping** The visual mapping is guided. This can mean that only possible values are allowed or that expressive and effective mappings are suggested. If the user ignores them, or interactions between multiple mappings occur, warnings may be issued.
	C-28	**Guidance for View Transformations** The (end) user is guided when viewing the visualisation. For example, he may be guided along an invisible route along a visual structure or he may have a restricted perspective in 3D space.
Means of guidance	**C-29**	**Guidance with a GUI** The guidance is given via a graphical user interface.

Name	Viz Designer + Lang.	Tableau + VizQL	TopBraid C. + UISPIN	Protovis	MDV + Cogz	SemViz	Cytoscape	RDFScape	Less + LeTl	Vispedia
Guidance for data filtering	(x)	(x)	-	-	-	-	x	x	(x)	x
Guidance for visual mapping	GUI	GUI	-	-	-	-	(GUI)	-	-	GUI
Guidance for view transformations	-	x	-	-	-	-	-	-	-	-

Table 4.8: Comparison of related visualisation systems and approaches – Guidance.

commercial desktop visualisation design suites for statistical data, but is rarely a feature of open web-based services for graph data.

Level of Ontology-Usage

While ontologies have been mentioned in the previous sections, the term *ontology-driven* has not yet been defined properly, which is important though, since it adds to the uniqueness of our approach. In this subsection, we define *ontology-driven* as a combined characteristic that includes some of the ontology-related criteria already mentioned before.

What adds ontology-driven visualisation to model-driven visualisation? The term *ontology-driven* has to be seen in the context of the term *model-driven*. What is it that ontology-driven adds to model-driven?

The OMG defines *model-driven* as »*an approach to software development whereby models are used as the primary source for documenting, analysing, designing, constructing, deploying and maintaining a system*« [Tru06]. Similarly, already Guarino [Gua98] defined and distinguished ontology-driven information systems and their development as the following, emphasising the temporal dimension: »*When the ontology is used by an IS [Information System] at run time, we speak of an 'ontology-driven IS' proper; when it is used at development time, we speak of 'ontology-driven IS development'* «. Following these definitions, we refer to an approach as *ontology-driven*, when ontologies are not only displayed and referenced, but when the whole system does heavily rely on ontologies at runtime.

Based on the OMG's definition of model-driven, Bull defines »*Model-driven visualisation*« as »*a model-based approach to view creation*« and »*a process for designing and customising interactive visualisations using principles from model-driven engineering*« [Bul08]. We build upon and extend these definitions to define »ontology-driven visualisation« as follows:

Definition 6 (Ontology-driven visualisation) *Ontology-driven visualisation describes an approach to visualisation design where ontologies are the primary models for analysing and constructing visualisations, as well as for storing and sharing visualisation knowledge and visual mapping definitions.*

Existing Ontology-Driven Approaches Table 4.9 summarises all criteria being concerned with the usage of ontologies that have been introduced before. We removed all tools from the list that do not make any use of ontologies.

All approaches are ontology-aware *(C-5)*, i. e., they can reference constructs in an ontology language and define ontology specific presentation information. Beyond this, for some of the visualisation approaches, we already noted the usage of ontologies as (a) a source of semantics that can be used to steer the visualisation and generate suggestions and (b) as a means to store and share visualisation knowledge. Further, more general, usage scenarios for ontologies in presentation or visualisation systems are possible, such as constructing presentation models or storing the presentation settings themselves based on ontology languages. These usage scenarios have been added here as well.

Semantics of the ontological data *(C-23)* are only considered for automating visualisation by CogZ, SemViz and Vispedia. However, they use ontology semantics on a very low level of ontological complexity. SemViz is the only approach that stores visualisation knowledge as ontologies *(C-25)*, but these ontologies have to be recreated by experts for each new domain and do not contain general rules and knowledge from empiric visualisation research.

TopBraid C. + UISPIN uses ontologies for constructing an HTML/SVG model and also stores RDF-based UISPIN presentation definitions. Even transformations can be directly formulated (as SPARQL 1.1 Update requests) on the level of ontological models. Also CogZ and SemViz both store mapping definitions in RDFS and additionally define a visual model for each type of graphic as an ontology. We described this as *other heavy use of ontologies* in the comparison table.

According to our definition, two visualisation approaches can be called actually driven by ontologies: SemViz and the approach of using the ontology mapping tool CogZ and MDV in combination. Vispedia could be referred to as partly driven by ontologies, mainly using them on the side of the source data. Also the TopBraid C. + UISPIN approach is fully ontology-driven, but is not a visualisation approach, since it does not support explicit visual mapping.

Name	TopBraid C. + UISPIN	MDV + Cogz	SemViz	Less + LeTl	Vispedia
Ontology-aware	x	x	x	x	(x)
Considers semantics of an ontology for visual mapping	n/a	(-)	(-)	n/a	(x)
Uses visualisation knowledge from ontology	n/a	-	x	n/a	-
Other heavy use of ontologies	x	x	x	-	-
Ontology-driven	x	x	x	-	(x)

Table 4.9: Comparison of related visualisation systems and approaches – Ontology usage.

4.1.3 Conclusion – What Is Still Missing?

Each of these approaches fulfills only a subset of the requirements, while none of them combines all the required characteristics. Fig. 4.7 gives an overview of the visualisation approaches that we analysed in the last section.

First, we have a group of visual mapping approaches, often offering guidance for visual mapping and sometimes also defining precise composition operators. Unfortunately, none of them is aware of ontologies, neither do they store their visualisation knowledge as ontologies nor do they use standards to enable the sharing of visual mappings. Here, the approach of Viz Designer and its languages ViZml and GPL is the one that is closest to our requirements.

Second, we have a group of ontology-driven approaches, some of them modelling composition, which do not explicitly store visual mappings, but can only be used for displaying ontologies in document style. Here, TopBraid C. + UISPIN comes close to our requirements, but visual mappings are not explicitly modelled in this approach.

From all approaches MDV + Cogz is the solution that is the most promising: It is ontology-driven, visual mappings can be modelled explicitly and composed to a certain extent. However, no guidance for the visualisation process is offered and some of the criteria are only very basically fulfilled. For example, visual mappings can only be done for concrete values, not for ranges of values. Furthermore, no composition of multiple visual structures can be defined.

With this thesis, we aim at filling the gap and find an approach to combine the four aspects *Visual mapping*, *Composability*, *Guidance* and *Ontology-driven* to enable ontology-driven guidance with the possibility to define composable visual mappings.

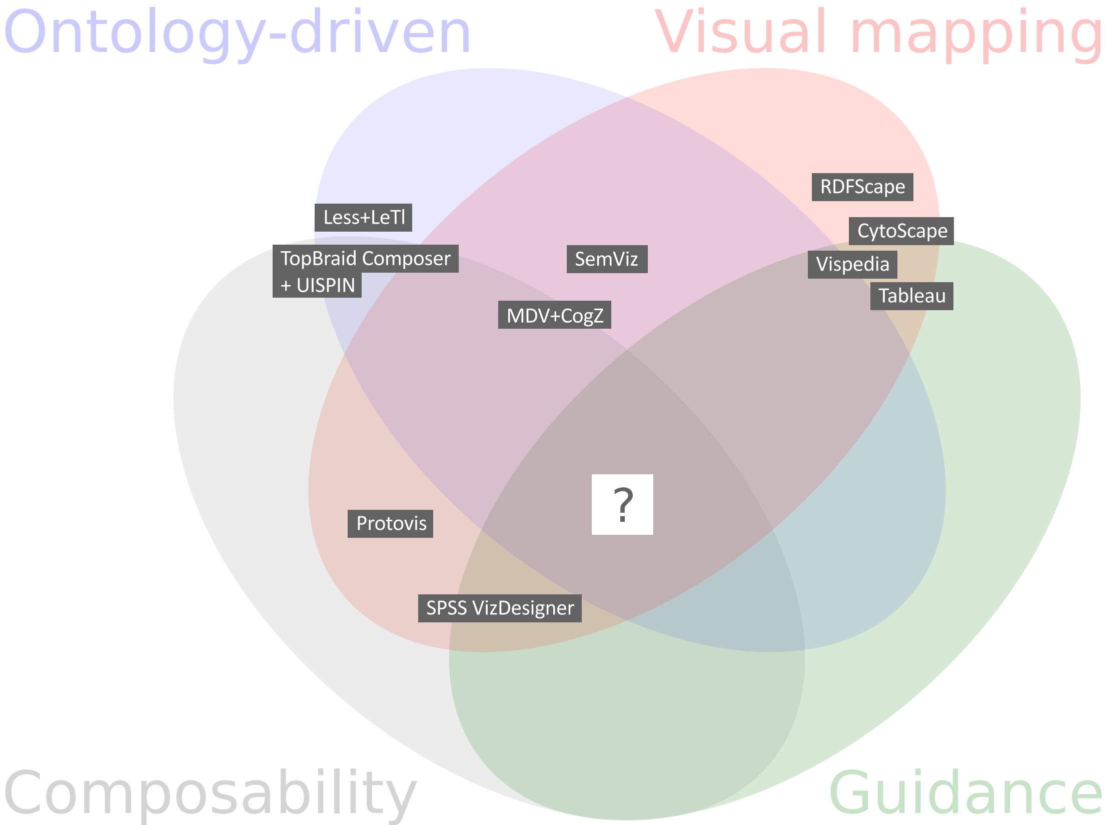


Figure 4.7: Overview of visualisation approaches. Existing work only covers a subset of the four aspects *Visual mapping*, *Composability*, *Guidance* and *Ontology-driven*, while there is a gap of solutions that address all of these aspects. What is more, most approaches only combine two aspects. An approach that enables ontology-driven guidance with the possibility to define composable visual mappings is missing. (A variant of this figure was shown in the introduction.)

4.2 Visualisation Languages

After having presented complete visualisation approaches in the last section, we now focus on *languages* for visualisation, many of them being part of the approaches introduced before. To explicitly define visual mappings for ontological data, we need to have a dedicated language for that purpose. A few visualisation languages exist, some of them very powerful and grounded in graphic algebras. However, none of them is targeted at ontology visualisation and aware of ontology specifics.

4.2.1 Short Overview of the Compared Languages

First, we briefly introduce each language. All visualisation languages will then be compared in detail in Sect. 4.2.2.

ViZml and GPL

Wilkinson [Wil05][11] describes a »grammar of graphics« by means of an algebra. Based on this graphic algebra, two languages are used by Wilkinson to allow for the description and automated production of graphics. Both were intended to be non-proprietary languages and introduced as open standards, however to the best of our knowledge, they are only used in SPSS. The first is the *Graphics Production Language* (GPL), which is an operational grammar-based language. The second is the *Visualisation Markup Language* (ViZml), a declarative markup language based on XML and defined by an XML schema. Both languages complement each other, having different benefits. While GPL can be used to specify graphics very concisely, ViZml is more extensible and can be easily checked by tools, based on its XML schema definition. GPL can also be generated from ViZml. ViZml is split into three parts that may be varied independently – the data, structure and style part. For styling, ViZml reuses CSS and SVG terms as much as possible, but it cannot directly be linked to CSS style sheets.

An example for a ViZml mapping is given in Listing 4.1, which shows a structure definition for a two-dimensional line chart, including axis labelling and colour settings. Also the concise definition of mappings to ranges of visual attributes is possible in ViZml, for example a data variable may be mapped to a range of visual (colour) values as shown in Listing 4.2.

```
<color variable='range' low='navy' high='red'>
```

Listing 4.2: ViZml – Example of mapping the data variable *range* to a colour range.

An example for a GPL program is given by Listing 4.3. GPL programs consist of statements, which are basically made up from a label defining the type of statement, and a set of functions. Graphic algebra expressions (using the cross, nest and blend operators) are also considered to be functions. The corresponding graphic is shown in Fig. 4.8.

[11] First published in 1999

```
<!-- Structure -->
<graph id="graph" cellStyle="graphStyle">
  <!-- Size -->
  <location method="fixed" part="right" value="100%"/>
  <location method="attach" part="right" target="container_39754"/>
  <coordinates>
    <dimension>
      <axis>
        <label purpose="auto" style="labelStyle">
          <descriptionGroup target="sourceVariable_39994">
            <description name="label"/>
          </descriptionGroup>
        </label>
        <majorTicks markStyle="majorTicksStyle3" style="majorTicksStyle"/>
      </axis>
    </dimension>
    <dimension>
      <axis>
        <label purpose="auto" style="labelStyle2">
          <text>Mean (</text>
          <descriptionGroup target="y">
            <description name="label"/>
          </descriptionGroup>
          <text>)</text>
        </label>
        <majorTicks markStyle="majorTicksStyle4" style="majorTicksStyle2"/>
      </axis>
    </dimension>
  </coordinates>
  <line positionModifier="stack" style="lineStyle">
    <binStatistic dimensions="1" gridType="square"/>
    <summaryStatistic convertIntervalToSingleValue="true" method="mean"/>
    <color id="color_39751" affect="main" cycle="cycle" missing="silver" scale="bipolarWhite"
         variable="y">
      <summaryStatistic method="mean"/>
    </color>
    <x variable="sourceVariable_39994"/>
    <y variable="y"/>
  </line>
</graph>
<container id="container_39754" clip="false">
```

Listing 4.1: ViZml (Visualisation Markup Language) – Example of a line chart (excerpt).

```
GGRAPH
  /GRAPHDATASET NAME="graphdataset" VARIABLES=schtyp MEAN(write)[name="MEAN_write"] female
  /GRAPHSPEC SOURCE=INLINE.
BEGIN GPL
SOURCE: s=userSource(id("graphdataset"))
DATA: schtyp=col(source(s), name("schtyp"), unit.category())
DATA: MEAN_write=col(source(s), name("MEAN_write"))
DATA: female=col(source(s), name("female"), unit.category())
COORD: rect(dim(1,2), cluster(3,0))
GUIDE: axis(dim(3), label("type of school"))
GUIDE: axis(dim(2), label("Mean write"), delta(5))
GUIDE: legend(aesthetic(aesthetic.color.interior), label("female"))
SCALE: linear(dim(2), min(40), max(60))
ELEMENT: interval(position(female*MEAN_write*schtyp), color.interior(female), shape.interior(
     shape.square))
END GPL.
```

Listing 4.3: GPL (Graphics Production Language) – Example of a grouped bar graph, taken from http://www.ats.ucla.edu/stat/spss/library/ggraph_examples.htm, accessed: 02.07.2015.

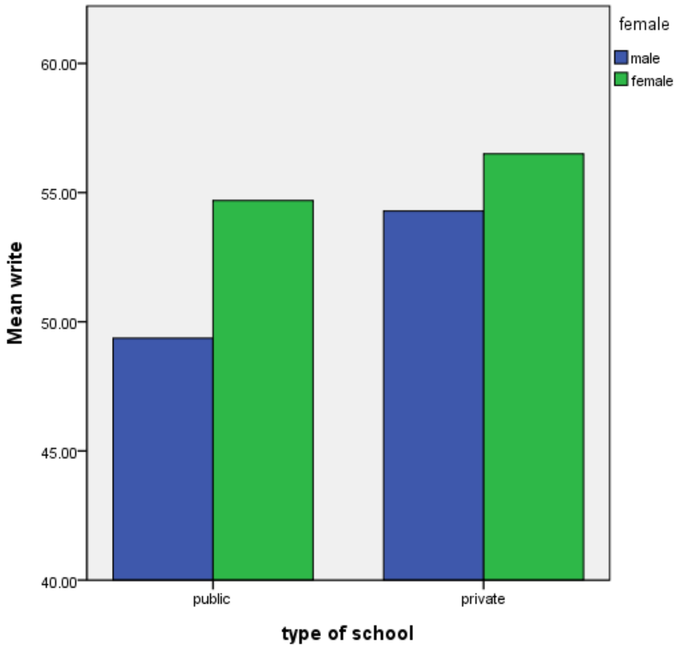
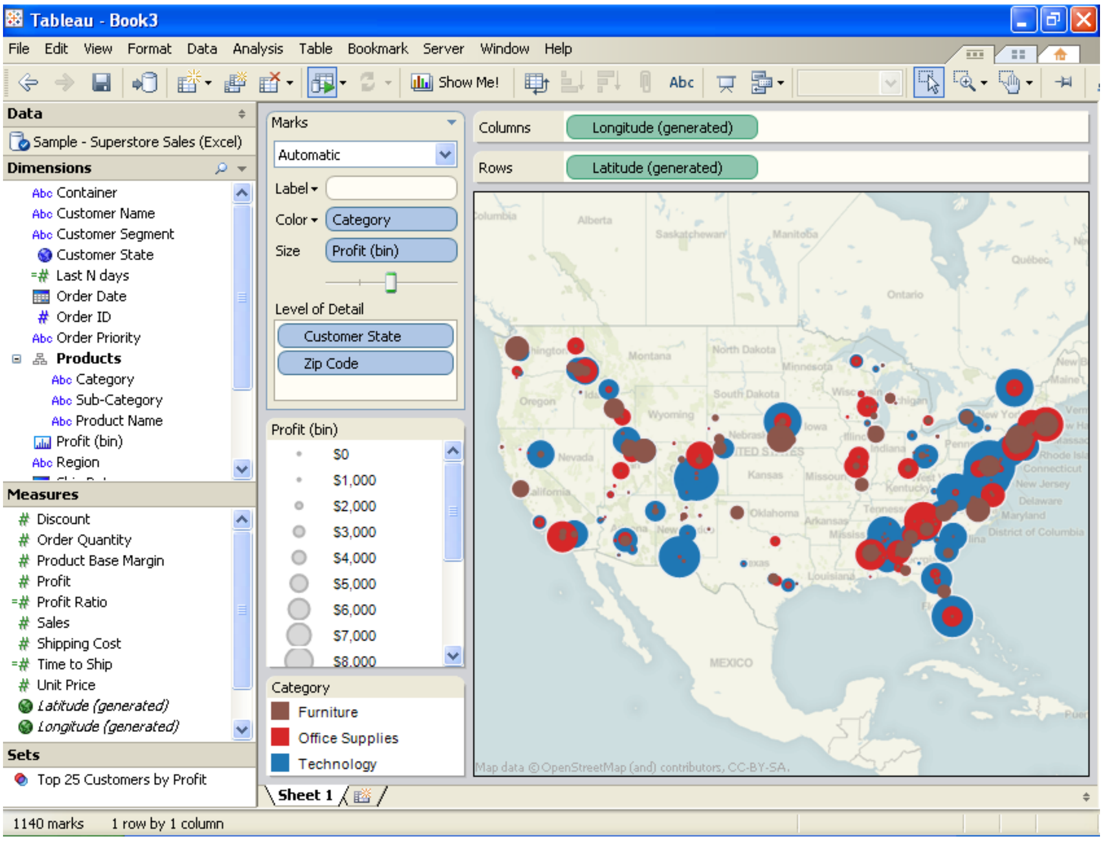


Figure 4.8: GPL – Example of a grouped bar graph (as described by Listing 4.3). In a grouped bar graph, bars are clustered in groups that emerge from examining an additional (nominal) attribute, in this example the sex of the participants.

VizQL

VizQL is a declarative language that extends the abilities of SQL by formatting and visualisation capabilities. This is inspired by the fact that SQL already offers some abilities to structure, sort and group data. The VizQL definitions are used by a query analyser to generate both visualisation settings and SQL, respectively MDX[12] for querying relational databases and OLAP servers [HSM07].

Mackinlay states in [MHS07] that VizQL is based on an improved algebra from the *A Presentation Tool* (APT) [Mac86a] and »*the ability to compile into database queries*«. He also compares Wilkinson's algebraic approach to VizQL and describes the main distinguishing feature as that it »*clearly describes the row and column structure of small multiple views of data*«. Both statements show that VizQL is, even more than GPL, targeted at the visualisation of tabular data. VizQL was first developed for the Polaris prototype and is now the basis of the commercial Tableau visualisation software (cf. Sect. 4.1.1 and Fig. 4.9).



Marks: Automatic

Label:

Color: Category

Size: Profit (bin)

Level of Detail: Customer State, Zip Code

Columns: Longitude (generated)

Rows: Latitude (generated)

Profit (bin): $0, $1,000, $2,000, $3,000, $4,000, $5,000, $6,000, $7,000, $8,000

Category: 🟫 Furniture, 🟥 Office Supplies, 🟦 Technology

1140 marks 1 row by 1 column

Figure 4.9: Tableau – User interface for VizQL. Although VizQL is readable, it is not intended to be written manually, but through a GUI.

Protovis

Protovis is a declarative visualisation language by Michael Bostock and Jeffrey Heer that has implementations as an embedded DSL in JavaScript [BH09] and Java [HB10]. It was designed to benefit from the advantages of declarative languages while still keeping an »intermediate level

[12] Multidimensional Expressions – Query language for OLAP databases, similar to SQL

of abstraction« to allow for flexible customisation of visualisations. This compromise between a declarativity and flexibility is achieved by using functions as mapping definitions, passed to other functions as (higher level) arguments. Also compare for Listing 4.4. The general declarative approach of the language allows for easily generating visualisations for other platforms. In the Java version, even the event model is »decoupled from the runtime platform« [HB10], in order to support cross-platform development.

When comparing visualisation languages in the next section, we refer to the JavaScript version of Protovis if not stated otherwise.

```
new pv.Panel()
    .width(150)
    .height(150)
  .add(pv.Bar)
    .data([1, 1.2, 1.7, 1.5, .7])
    .bottom(2)
    .width(20)
    .height(function(d) d * 80)
    .left(function() this.index * 25 + 2)
    .strokeStyle(function(d) (d > 1) ? "red" : "black")
  .root.render();
```

Listing 4.4: Protovis – Example of a bar chart. In Protovis, visualisation settings are specified by chaining function calls one after another, each call working on the return value of the previous call (method chaining). By this means, first, a panel is described and then a bar is added to this panel. The bar in turns is passed an array of source data values, static presentation values for defining bottom position and width, but also dynamic values as functions calculating height, left position and stroke style (colour) from the given data. Finally, the render action is triggered just like the declarative settings have been done before.

4.2.2 Detailed Comparison by Language Criteria

After having briefly introduced each language, we now compare them according to several criteria we consider important for our own ontology visualisation language[13].

Supported Data Models

Since the visualisation languages we found mainly have a background of visualisation for statistics, where we have a lot of homogeneous data that can well be stored in relational databases, the **tabular data model** is the kind of data model that is supported by all languages *(LC-1)*. ViZmL and VizQL have been built to be used with relational databases. Protovis has functions for handling tabular data, and also some functions that aim at hierarchical or graph data structures (for example, setting the link degree or choosing between various force directed layouts). However, the other visualisation languages do not **support graphs** *(LC-2)* to a greater extent; no graph file formats can be referenced, implying that **RDF** is neither supported *(LC-3)*, and few functions exist that are specific to graphs. The focus is clearly on the needs of statistics, although some means to display a node-link-diagram are available.

Addressable Language Constructs

We further ask, whether constructs of (a potential) **schema** of the source data **can be referenced** within mapping definitions *(LC-4)*, which is possible with all languages, except Protovis, to the extent that columns of a database schema can explicitly be mapped to visual means. Protovis does not refer to a database directly, but uses internal JavaScript arrays for storing

[13] The corresponding requirements will be listed in detail in Sect. 7.1.

Supported data models

LC-1 Tabular data model supported The language can reference source data from a tabular data model.

LC-2 Graphs supported The language can reference data that forms a network or graph data specific settings are possible.

LC-3 RDF supported RDF or the languages build on top of it (RDFS, OWL) are supported.

Addressable constructs

LC-4 Schema/types addressable The language can reference the data's schema within display definitions. For graphs, the type of edges can be referenced at least.

LC-5 Ontology constructs addressable The language can reference to ontology terms (in any formalism). Switching on/off inference is optional.

LC-6 Terminological ontology relations (T-Box) addressable Specific representation of statements on the relation between classes and properties – T-Box (terminological component). It is possible to reference these relations explicitly in the display definition.

Name	ViZml	GPL	VizQL	Protovis (JS)
Tabular data model supported	x	x	x	x
Graphs supported	(x)	(x)	(x)	x
RDF supported	-	-	-	-
Schema/types addressable	x	x	x	-
Ontology constructs addressable	-	-	-	-

Table 4.10: Comparison of visualisation languages – Supported data models and referencable constructs.

data. However, none of these languages can **reference ontological constructs** *(LC-5)* or has specific features for handling ontologies, neither on A-Box nor T-Box level. The criterion of **T-Box support** *(LC-6)* will be used later, in the next section on RDF display languages, and is listed here only for the sake of completeness.

Interactivity

Except for GPL, all visualisation languages have special terms for conveniently defining **simple interaction tasks** *(LC-8)*: In ViZml, there are language constructs for defining some interactions, however none of them is configurable via the Viz Designer UI. The whitepaper on Tableau and VizQL [HSM07, p. 19] further states that interactive visualisations can be »*described by a simple* [. . .] *VizQL statement that Tableau's interpreter translates into an interactive information visualization*«. Among these interactions are *linking and brushing*[14] and *tool-tips*. Also in Protovis mouse events can be mapped to pan and zoom behaviours.

Language Paradigm

All languages we examined can be referred to as Domain Specific Languages (DSLs) being designed to serve a purpose that is specific to a certain domain – in our case visualisation respectively the display of RDF data. However, we can further classify these DSLs into **declarative** *(LC-9)*, **imperative** *(LC-10)* and hybrid languages [Wu07]. For the group of visualisation languages, all languages we inspected are declarative, except for GPL, which has

[14] When items are selected in one of multiple views, all views show the selection [CMS09].

Visual Mapping — **LC-7 Visual Mapping (not only presentation)** Visual mappings can be explicitly defined.[1)]

Interactivity — **LC-8 Interaction describeable** Interactions with the visualisation can be defined. E. g., »What happens on select?«.

Name	ViZml	GPL	VizQL	Protovis (JS)
Interaction describable	x	-	x	x

1) Applies to all visualisation languages, according to our definition and is listed here only for completeness. It will be needed for comparing RDF display languages in the next section.

Table 4.11: Comparison of visualisation languages – Interactivity.

Language paradigm — **LC-9 Declarative** The language describes WHAT should happen as opposed to describing HOW exactly it should happen.

LC-10 Imperative The opposite of declarative style describing exactly HOW to do a calculation in terms of assignments and control flow statements.

LC-11 Templates Templates are used to generate code or more specific models.[1)]

Name	ViZml	GPL	VizQL	Protovis (JS)
Declarative	x	(x)	x	x

1) Applies to imperative languages and is listed here only for the sake of completeness. It will be used to further characterise RDF display languages in the next section.

Table 4.12: Comparison of visualisation languages – Language paradigm.

the DO control statement for iterating and conditional execution. However, we will see in Sect. 4.3.2 that for RDF display languages both paradigms are used.

Differences between the four visualisation languages exist in the way they are implemented. While ViZml, GPL and VizQL are defined as completely new languages (although ViZml is based on XML and VizQL is oriented at SQL), the designers of Protovis chose the solution of a DSL, embedded into JavaScript [BH09]. Also a Java-embedded version was built [HB10]. A discussion of the advantages and disadvantages of declarative vs. imperative languages will be given in Sect. 7.1, when discussing our requirements for an RDFS/OWL visualisation language.

Platform Independence and Visual Structure Independence

The criteria of platform and visual structure independence relate to the criteria of variability *(C-15, C-17)* that we discussed in Sect. 4.1.2. To allow for variation of platforms and visual structures, the language used for storing display definitions needs to be independent of platforms respectively specific visual structures. While all visualisation languages are **platform-independent** *(LC-12)* and all of them support **multiple visual structures** *(LC-14)*, only VizQL allows for specifying visual mappings **independently from the type of the chosen**

Platforms

LC-12 Platform independence The mapping/view definition (at least part of it) is done without referring to a specific platform, i. e., there are some settings that are valuable for all platforms.

Visual Structures

LC-13 Visual structure independence The view/mapping definition (at least part of it) is done without referring to a specific visual structure.

LC-14 Multiple visual structures/paradigms Multiple Visual structures can be described. This includes the explicit description of visual structures and the construction by general purposes languages.

Name	ViZml	GPL	VizQL	Protovis (JS)
Platform independence	x	x	x	x
Multiple visual structures/paradigms	x	x	x	x
Visual structure independence	(x)	-	x	(x)

Table 4.13: Comparison of visualisation languages – Platform and visual structure independence.

main visual structure[15] *(LC-13)*. In ViZml, style sheets and visual structure are separated, so that they can be varied independently. However, other mappings than styles are directly inter-linked with the (main) visual structure, so that changing this structure will require redefining these other mappings. Also Protovis »*reuses the semantics of* [..] *properties as much as possible*« [HB10] to support consistency. That means mappings to visual attributes can sometimes be reused, when changing the visual structure.

Shareability and Reusability

Although every mapping definition might be **shared** with others, there are means of supporting this already by a visualisation language *(LC-15)*. Examples for this are Protovis and ViZml. Protovis intentionally describes visualisations simply as JavaScript, such that »*visualisations become open source*« [BH09] and can be reused and extended by others easily. ViZml lowers the barrier to share and reuse mapping definitions by using XML that can be easily processed on many platforms and for which a lot of tooling exists. Unlike RDF display languages, none of the visualisation languages uses **URIs to uniquely identify created mappings on the web** *(LC-16)*. This could help when multiple users share mappings, especially when already the source data can be found on the web at a URI. Shareability implies *reusability*, which we do not list separately.

Extensibility

Extensibility refers to whether or not existing **mappings can be extended** *(LC-17)*. ViZml is based on XML (the Extensible Markup Language) and, therefore, is generally extensible. The ViZml XSD was intentionally designed to allow for extensions of the schema. For situations, where changing the schema is impractical, another extension mechanism is offered by the »Parameter and Method« nodes that allow for a very flexible generalisation. In GPL and VizQL, there is no support to extend the mappings easily, such that new mappings could be built on existing ones and inconsistency is avoided when mappings need to be changed. Protovis uses »mix-in functionality« to allow for prototype inheritance.

[15] In VizQL, a main visual structure is chosen by selecting the type of graphic

Shareability	**LC-15 Shareability supported by language** View/mapping definitions can be shared among various users on different platforms visualising different instance data. Sharing is supported by some mechanism, e. g., URIs. **LC-16 URIs for each view/mapping** Each mapping/view gets a URI.
Extensibility	**LC-17 Extensibility of views/mappings** Existing view/mapping definitions can be extended, e. g., by inheritance.
Composability	**LC-18 Composability of views/mappings** Explicitly defined views/mappings can be combined by third parties to build more complex ones. (Same as *C-18*)

Name	ViZml	GPL	VizQL	Protovis (JS)
Shareability supported by language	x	-	-	x
– URIs for each view/mapping	-	-	-	-
Extensibility of views/mappings	x	-	-	x
Composability of views/mappings	x	x	-	x

Table 4.14: Comparison of visualisation languages – Extensibility, shareability and composability.

Composability of Visual Mappings and Views

We refer here specifically to the composability of mapping definitions and views *(LC-18)*, which is equivalent to the criterion *C-18* from the comparison of visualisation approaches.

Although all the languages allow for applying multiple visual mappings, the composition of mappings that have been stored earlier is supported differently well. Therefore, the reuse of existing mappings is sometimes difficult: Protovis allows for storing partial mappings, which are nested function calls, in variables. This way, they can be referenced and composed, but from outside the scope of the Protovis program, it will be harder to reference them, compared to UISPIN components, which have a URI (Sect. 4.3.1). The same applies to GPL [Wil05], which allows for storing visualisation settings in variables for later use. In ViZml, it is possible to create two graphics and display them next to each other. While this kind of composition is supported, no other complex compositions seem to be available. For his graphic algebra, Wilkinson defines three operators, *cross (*)*, *nest (/)*, and *blend (+)*, that work on sets of variables. However, these are not graphic-level composition operators but data-level operators. They are closely related to operators such as the JOIN operator from relational algebra and allow for writing multi-dimensional algebraic expressions for preparing data sets.

A different criterion, which has not been examined here, is, whether or not the composability of definitions is constrained in a reasonable way to prevent the creation of mappings that are syntactically valid, but impossible to realise. We return to the aspect of composition in Sect. 8.2.

Construction of Editors

To decide whether one of the visualisation languages could be extended towards ontology visualisation, it is interesting to know how easily an editor for visual mappings can be built or if an existing editor could be extended.

None of the languages we inspected offers an **open-source editor** that could be reused *(LC-20)*. ViZml, GPL and VizQL are used within commercial software and hence do neither have a free API available. In the case of Protovis, an implementation is freely available under the BSD License. However, also here, no tooling for editing mappings exists.

Another question related to the construction of editors is, whether the language is described

Editor construction

LC-19 Restrictive tool-usable schema The language definition is directly usable by tools to restrict instance creation. E. g., an XSD offers constraints that XML editors can use to allow only schema compliant XML code.

LC-20 Open source editor An editor for the mapping definition is available open source.

Familiar to users

LC-21 Familiar to users If complex queries or other instructions need to be defined, the language should be familiar to the users and the same as for other parts of the configuration if possible. (*Criterion oriented at [Bul08]*)

Name	ViZml	GPL	VizQL	Protovis (JS)
Restrictive tool-usable schema	x	(x)	-	-
Open-source editor	-	-	-	-
Familiar to users	x	-	(x)	(x)

Table 4.15: Comparison of visualisation languages – Construction of editors and familiarity to users.

with a formal and **restrictive tool-usable schema** *(LC-19)* that is strict enough to allow for automatically deriving an editor. In the case of ViZml, which is described via an XML schema, this is true at least partially and for the syntax. Tools can easily check ViZml code against the schema. Also GPL is defined with an EBNF grammar, however, this grammar does not seem to be available on the web. For VizQL and Protovis, there exists no such schema.

Familiarity to users

Except for GPL, all visualisation languages build upon existing knowledge of visualisation authors in using common languages *(LC-21)*. ViZml is familiar to users who are used to handle XML; VizQL builds on existing SQL knowledge and Protovis aims at users with a web background already having some JavaScript skills.

4.2.3 Conclusion – What Is Still Missing?

Although some visualisation languages exist, none of them can be used to define visual mappings directly between ontological terms and visual means. So, for example, ViZml and GPL are advanced visualisation languages with a profound algebra-founded concept for composition. However, they do not allow for referencing ontology constructs and are targeted at visualising homogeneous tabular data in first place.

4.3 RDF Presentation Languages

After having compared *visualisation languages* in the last section, we look at existing *RDF presentation languages*. We also look at related non-RDF languages that have been developed for displaying and styling general graphs and could also be applied to RDF graphs. As we will see, RDF presentation languages are not a subset of visualisation languages.

4.3.1 Short Overview of the Compared Languages

First, we briefly introduce the RDF presentation languages. All languages are then compared in detail in Sect. 4.3.2.

Graph Style Sheets

Graph Style Sheets (GSS) is an RDF vocabulary that can be used to modify the visual appearance of a node-link diagram with style rules. Unlike Fresnel, GSS basically do not transform the graph, but style it. That means, the data's graph structure is (directly) transferred into a visual structure, i. e., a node-link-diagram. However, there are exceptions to this, e. g., it is possible to combine the node-link representation with tables, representing statements that belong to one object in a more compact way (cf. Listing 4.5). A second exception is the complete removal of elements from display, thereby changing the node-link structure.

GSS is oriented at CSS, often accepting the same values, however it does not actually integrate and reuse it. For example, cascading and weighting is used to allow the combination of multiple style sheets. Similar to CSS, a style rule consists of a left-hand side, called *selector*, and a right-hand side that holds the styling advices. GSS are used in the RDF tool IsaViz [Pie07], which also provides a graphical editor for GSS.

```
[ rdf:type gss:Property ;
  gss:uriStartsWith d: ;
  gss:display gss:None
] .

[ rdf:type gss:Resource ;
  gss:style :presidentStyle ;
  gss:subjectOfStatement [ gss:predicate rdf:type ]
  gss:layout gss:Table ;
  gss:sortPropertiesBy gss:Namespace ;
] .

:presidentStyle gss:shape gss:RoundRectangle .
```

Listing 4.5: Example GSS style removing all properties from namespace d and representing all resources that have an rdf:type assigned as round rectangles with their properties being represented as a table

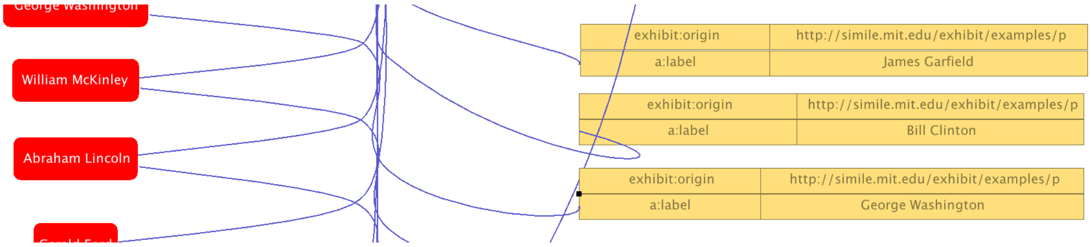


exhibit:origin	http://simile.mit.edu/exhibit/examples/p
a:label	James Garfield

exhibit:origin	http://simile.mit.edu/exhibit/examples/p
a:label	Bill Clinton

exhibit:origin	http://simile.mit.edu/exhibit/examples/p
a:label	George Washington

Figure 4.10: Result of the GSS styling given in Listing 4.5 (screenshot from IsaViz).

VPOET

VPOET [RCC09] enables web designers to embed macros for displaying and editing RDF data into web pages without requiring expert knowledge on Semantic Web languages. However, ease of use for the web designers' community is given priority over advanced functionality.

Fresnel

Fresnel was developed as a »Presentation Vocabulary for RDF« [BLP05] in order to offer a common language to define reusable presentation information for RDF. This became necessary, because many projects started developing proprietary languages for this purpose on their own. The authors argue that there is no reason why the benefits of semantic web languages (i. e., unique identifiers, shareability, inference) should not also be used for presentation information as well; therefore, they defined the declarative language Fresnel on top of RDFS/OWL.

Fresnel distinguishes three steps in a presentation process – *selecting* what to present, *structuring* the items to be presented, and finally *formatting* them. It does not specify how to render them. Selection and structuring is handled by the Fresnel *Lenses*. With the lens construct, Fresnel introduces an explicit definition of a *view*. Furthermore, Fresnel offers a means to loosely connect the view to a set of resources by means of various selectors. Formatting, the assignment of layout information is done with the help of Fresnel *Formats*, which can, for example, attach CSS-styles to resources.

Lenses and Formats need to define selection expressions, e. g., to specify the properties to be shown by a Lens. This can be done in Fresnel via a simple URI reference, or – for increasingly complex selections – via the languages **Fresnel Selector Language** (FSL) and SPARQL. FSL [Pie05] is a language to describe traversal paths through RDF graphs (with limited expressiveness) and allows for more compact selection expressions than SPARQL does. For example, the FSL expression »foaf:knows/foaf:Person/foaf:name« selects the foaf:name property of all instances of foaf:Person known by a given resource.

The most important characteristics of Fresnel are its platform- and visual paradigm-independence. We discuss this in detail in Sect. 4.3.2.

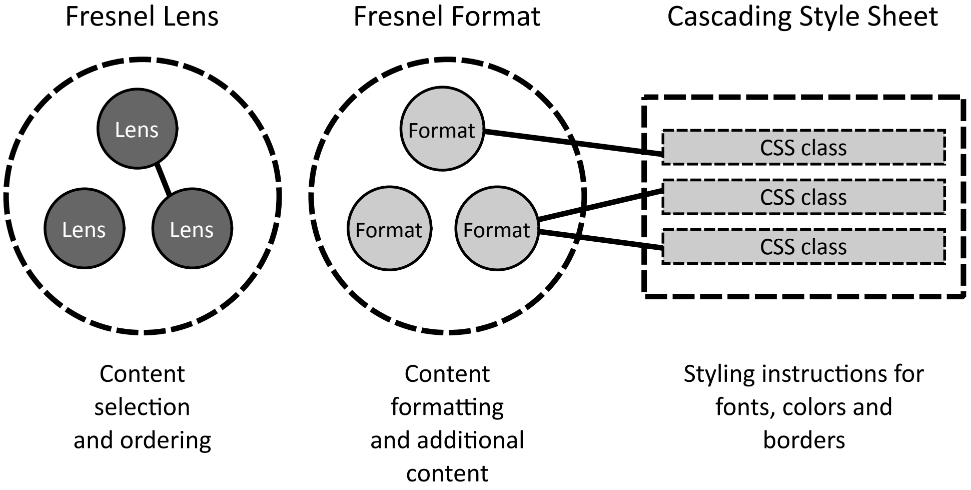

Fresnel Lens
Fresnel Format
Cascading Style Sheet
Lens
Lens
Lens
Format
Format
Format
CSS class
CSS class
CSS class
Content selection and ordering
Content formatting and additional content
Styling instructions for fonts, colors and borders

Figure 4.11: Foundational concepts of Fresnel. Redrawn after Pietriga et al. [PBKL06b].

Fresnel Application Fresnel has been used in the applications listed in Table 4.16. We noted whether Fresnel or one of its extensions is used. Furthermore, we distinguish applications that

Application	extended	by Fresnel inventors
Longwell[16]	no	yes
IsaViz (Sect. 4.3.1)	no	yes
Geonames Browser [PBKL06a]	yes (2D)	yes
Hyena [Rau05]	yes (editing)	no
Lena[17]	no	no
DBpedia Mobile[18]	no	yes
Arago[19]	no	no

Table 4.16: Applications using Fresnel.

have been worked on by the Fresnel inventors themselves. Although Fresnel was only used by a few external authors, this is more frequent than for any other RDF display or visualisation language we inspected, which have not been reused in other contexts to our knowledge.

Fresnel2D

Fresnel2D is an example of an extension vocabulary for Fresnel for the 2D-layout of RDF data. It was used in the »Geonames Browser« to layout geographical locations on a map using longitude and latitude properties. It declaratively defines a mapping from the geographical data to the x- and y-positions of the map by defining the range of longitude (-180°–180°) and latitude (-90°–90°) that should be mapped to the whole range of the available x- and y-positions. Fresnel2D was introduced as work in progress on ISCW 2006 [PBKL06a], however, we could not find later publication on the extension.

RDF Editing Metamodel (REMM)

The RDF Editing Metamodel introduced by Rauschmayer [Rau10] extends Fresnel with editing-specific features. It is used by the Hyena application to provide form-based editing for RDF. The idea is to use Fresnel to define an editing view. Because Fresnel is not specialised on editing, REMM extends it by offering constructs to define defaults and editing profiles for RDF. For a given property, these profiles can define possible types that newly created objects should have. REMM also extends Fresnels ability to display value hints (e. g., `:someProperty fresnel:value fresnel:image.`) to enable the suggestion of widgets that are suitable for editing values of the property (e. g., `:someProperty remm:editHint remm:EditCheckBox.`). Another example is given in Listing 4.6.

```
[ a fresnel:PropertyDescription ;
      fresnel:property ex:genre ;
      remm:propertyRange ex:Genre ;
      fresnel:use [
             a fresnel:Format ;
                   remm:editHint [
                         a remm:EditInstanceCollection
                   ]
      ]
]
```

Listing 4.6: REMM Example – Fresnel and REMM expressions are mixed to define that a collection widget showing instances of the class Genre should be displayed for editing.

[16] Longwell browser. SIMILE project, MIT and CSAIL. http://simile.mit.edu/longwell/

[17] A Fresnel LEns based RDF/Linked Data NAvigator with SPARQL selector support [KFS08]. University of Koblenz.

[18] DBpedia Mobile: A Location-Aware Semantic Web Client [BB08]. Freie Universität Berlin.

[19] From Graph to GUI: Displaying RDF Data from the Web with Arago [GH05]. Université de Fribourg.

Tal4RDF

Tal4RDF [Cha09] is a pragmatic, lightweight template language for RDF presentation that aims at being simpler than Fresnel and thereby more feasible for rapid prototyping. It is based on Zopes TAL[20] and written in Python. Special tags are integrated into the XML-tags of the original document, which remains a valid document in the »host« format. This has the benefit that the host language can still be statically type checked (assuming the host language is also XML-based). Language features include condition-tags (if), repeat-tags (looping) and the bundling of similar properties using »OR«. The developers see it as a benefit that they directly aim at rendering the presentation format. The downside of this is, however, that although all formats could be rendered (such as SVG or JavaScript), a mapping of RDF-properties to graphical means, depending on their values, cannot explicitly be modelled with the language. Also, complex queries cannot separately be defined, but have to be combined with the template.

LESS Template Language

The LESS Template Language (LeTL) [ADD10] is the template language used in LESS (Sect. 4.1.1) and inherits from the Smarty Template Language[21] for PHP.

Users can create output fragments with placeholders for RDF-data using LeTL. These views can then be shared and also taken as a prototype for new views. The authors state that they could also have included Tal4Rdf or Fresnel, but wanted to keep it simple. Future integration of these languages has been considered. In contrast to Tal4RDF, query and template are stored separately. The template then references variables from the (SPARQL) query result. Also LeTL does not support visual mapping or interaction explicitly.

XSLT-Based Approaches

At first glance, it might seem suitable to use XSLT style sheets for transforming RDF data into a human readable format. However, RDF/XML is only one out of many serialisations for RDF. Others are N3 and Turtle. Moreover, even the XML serialisation cannot be processed directly with plain XSLT+XPATH, since it represents only one possibility of stating the same facts in RDF. Although the RDF/XML describes a hierarchy, this hierarchy does not necessarily represent a useful structure for human representation. In [Ste03] the following further shortcoming of using XSLT+XPATH for RDF/OWL are given: First, owl:imports statements cannot be handled, so that only a single file can be considered. Second, it is not possible to process implicit knowledge because XSLT is not aware of inference.

To process not only the serialisation structure but the RDF graph behind it, extensions of XSLT and XSLT-influenced approaches have been developed. Xenon [QK05] and RDF Twig [Wal03] belong to this category of languages. To access the data graph, these approaches use path traversal query languages such as RDFPath [Ste03], which is similar to XPATH but considers RDF specifics. In [FLB$^+$06], Furche et al. mention other similar languages. However, as with RDFPath, there seems to be no further work on them. These approaches have the characteristic that they are procedural and, therefore, are bound to a concrete output format. It is also not possible to define display paradigm-agnostic presentation information.

OWL-PL

OWL-PL [BH10] is not an XSLT-based but an XSLT-inspired language for transforming RDF/OWL into XHTML. It aims at producing both machine and human readable data. The language provides the definition of default views. It further defines an order of precendence between default views, views coupled to classes, views coupled to instances and user-defined

[20] http://wiki.zope.org/ZPT/TAL/, accessed: 02.07.2015.

[21] Smarty Template Language. http://www.smarty.net/, accessed: 02.07.2015.

view settings, which supersede all other settings. Connecting views (format files) and resources is done with the help of vocabulary from a »formatting ontology«. The language comprises constructs for conditions (if, when, otherwise) and for looping (for-each, for-all, for-others). Views can be reused similar to the reuse that XSLT-templates offer. Inference is not handled by the language, but expected to be solved by ontology mapping in the data layer.

STOOG Graph Stylesheets

STOOG is a style sheet language[22] for graphs. In contrast to the GSS, it is not explicitly targeted at RDF graphs, but at general *multivariate graphs* with attributed nodes and edges and optional clustering information. Although STOOG is not ontology-aware, we added the language to our comparison for its close relation to CSS. As with GSS, the cascading mechanism known from CSS is supported, but STOOG is close to CSS with respect to syntax as well. Additionally, the language has slightly more direct visual mapping abilities than GSS.

UISPIN

UISPIN, also called SPARQL Web Pages (SWP)[23] [Knu10], is a declarative RDF-based UI-framework used in TopBraid Composer[24] (Sect. 4.1.1) to define HTML and SVG views of ontological data. UISPIN builds on SPIN, which we introduced in Sect. 2.2.7 as a means to attach (SPARQL-based) constraints and rules to RDFS classes.

UISPIN offers properties that can be used to link RDFS resources with descriptions of the UI (cf. Fig. 4.12). The UI components are declaratively described and contain SPARQL queries to dynamically insert data. The common UI-tags, specific XHTML-tags and SPARQL expressions are all together transparently transformed into an RDF representation for storage, allowing for a unified handling of all three – data, presentation information, and queries. Static type checking for HTML and SVG is possible, since their schemata were completely re-modelled in RDF. This is also the prerequisite, if a new target language should be used with UISPIN. It is generally possible to build visualisations with UISPIN, since also charts, maps and other diagrams can be included using UISPIN *Charts*[25]. However, only out-of-the-box Google widgets can be embedded. That means, it is not possible to actually model or interactively configure the visualisation. A drawback of UISPIN is that its textual notation is hardly human-readable, because like all SPIN-based expressions the templates and functions result in complex RDF list structures. That means UISPIN definitions always require an appropriate UI for presentation and editing.

4.3.2 Detailed Comparison by Language Criteria

Having introduced related languages for RDF presentation, we now compare these languages in detail. We use the language criteria *LC-1 – LC-21* that we already introduced for visualisation languages in Sect. 4.2.2. Further criteria that are specific to RDF display languages are added and used for additional comparisons in Sect. 4.3.3.

Supported Data Model and Referenceable Source Language Constructs

All languages can **reference terms from some kind of schema** *(LC-4)*. All of them are **aware of ontology terms** *(LC-5)* in an RDF-based language *(LC-3)*, except for STOOG.

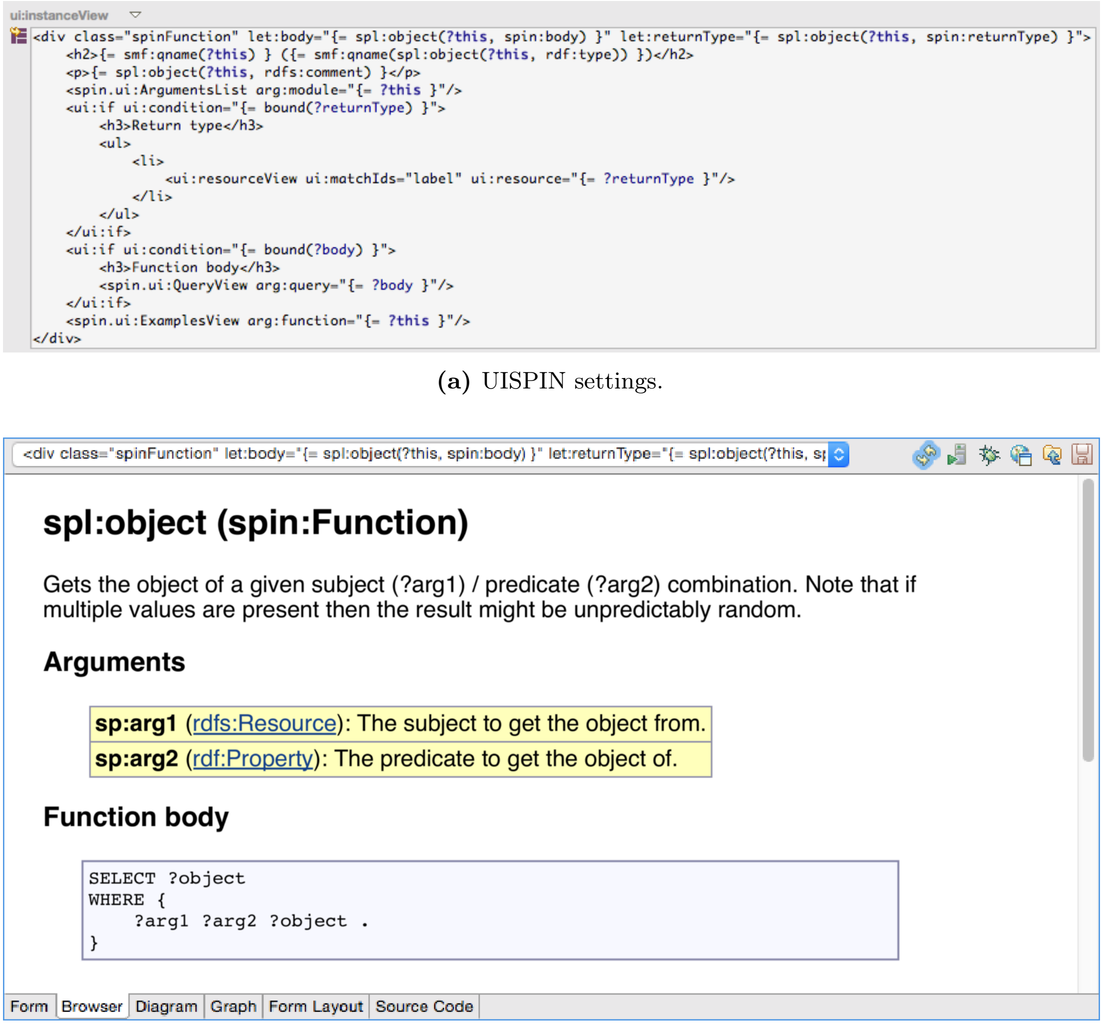
[22] Although STOOG is also a toolkit, we did not consider these aspects in our comparison, since it lacks an available implementation for further testing.

[23] http://uispin.org/, accessed: 02.07.2015.

[24] http://www.topquadrant.com/topbraid/, accessed: 02.07.2015.

[25] http://uispin.org/charts.html, accessed: 02.07.2015.

[26] Example taken from the blog »Composing the Semantic Web« by Holger Knublauch. http://composing-the-semantic-web.blogspot.com, accessed: 12.01.2016.



```
<div class="spinFunction" let:body="{= spl:object(?this, spin:body) }" let:returnType="{= spl:object(?this, spin:returnType) }">
    <h2>{= smf:qname(?this) } ({= smf:qname(spl:object(?this, rdf:type)) })</h2>
    <p>{= spl:object(?this, rdfs:comment) }</p>
    <spin.ui:ArgumentsList arg:module="{= ?this }"/>
    <ui:if ui:condition="{= bound(?returnType) }">
        <h3>Return type</h3>
        <ul>
            <li>
                <ui:resourceView ui:matchIds="label" ui:resource="{= ?returnType }"/>
            </li>
        </ul>
    </ui:if>
    <ui:if ui:condition="{= bound(?body) }">
        <h3>Function body</h3>
        <spin.ui:QueryView arg:query="{= ?body }"/>
    </ui:if>
    <spin.ui:ExamplesView arg:function="{= ?this }"/>
</div>
```

(a) UISPIN settings.

```
<div class="spinFunction" let:body="{= spl:object(?this, spin:body) }" let:returnType="{= spl:object(?this, s
```

spl:object (spin:Function)

Gets the object of a given subject (?arg1) / predicate (?arg2) combination. Note that if multiple values are present then the result might be unpredictably random.

Arguments

sp:arg1 (rdfs:Resource): The subject to get the object from.
sp:arg2 (rdf:Property): The predicate to get the object of.

Function body

```
SELECT ?object
WHERE {
    ?arg1 ?arg2 ?object .
}
```

(b) Rendered HTML.

Figure 4.12: UISPIN HTML display settings (a) and the corresponding rendered HTML view (b) as displayed by a browser. The display settings are attached to the class SPINFunction via an »instance view«. This means the settings are applied to instances of the class, but not the class itself.[26]

Name	GSS	VPOET	Fresnel	REMM	Tal4Rdf	LeTL	OWL-PL	STOOG	UISPIN
Schema/types referenceable	x	x	x	x	x	x	x	x	x
Ontology constructs referenceable	x	x	x	x	x	x	x	-	x
T-Box referenceable	(-)	-	(-)	(-)	(-)	(-)	-	-	(-)

Table 4.17: Comparison of RDF display languages – Supported data models and referencable constructs.

STOOG works on general multivariate graphs with nodes and edges that may have additional attributes.

Therefore, we have a closer look at the exact terms from a schema or ontology that can be referenced directly with the language. The **T-Box**, i. e., relations between classes and relations, cannot be referenced directly in display definitions in any of the languages *(LC-6)*. Nevertheless, in some languages mapping relations that exist indirectly due to existential or universal restrictions can indirectly be achieved by using a selector language. For example, Fresnel and REMM support the Fresnel Selector Language (FSL) and also Tal4RDF has its own path-based selector language. In others, SPARQL can be used for this purpose (UISPIN and LeTL).

Visual Mapping

We defined *visual mapping (LC-7)* as the mapping of relations and values in the data to visual means (Sect. 2.1.5), including dynamic, value-dependent mappings. This means, mappings can be formally described on the level of RDFS properties and visual means.

None of the RDF presentation languages is capable of defining such mappings. As an example, it is not possible in any of the languages to define mappings from RDF properties to visual attributes such that each value of the property is implicitly mapped to some visual attribute value. Fig. 4.13 illustrates the difference between mapping on a property level and mapping single instances to visual values.

With GSS, as already described above, at least complete graphic representations can be chosen as »layout« with two options: »Table« and »Node and Arc«. Visualisations can also be created with the UISPIN extensions *UISPIN Charts*, but for this, ready-to-use visualisation components are used, which do not allow for composition and reasoning about further possible additional visual mappings, because the semantics of their configurable »slots« are not formally described. STOOG style sheets allow for defining more complex mappings than only one-to-one value mappings by expressions such as

$$@f\_math.max(@self.weight/15 \ast 255) + 13$$

However, again the mapping between a source data variable and a target visual attribute is not explicitly modelled, leading to the same problems mentioned above.

Interactivity

Interaction with the graphic representation *(LC-8)* is possible with all languages except GSS, but often in a very limited way. In other cases interactions cannot explicitly be modelled, but can only be achieved by means of HTML or a general-purpose language such as JavaScript. Fresnel and STOOG have a few concepts to model interaction, similar to the pseudo-selector classes of CSS.

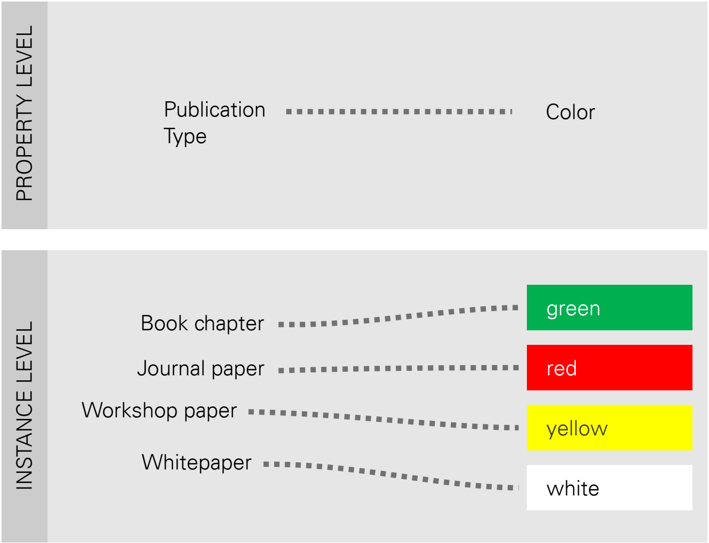


Figure 4.13: Visual mapping on the level of properties and classes (top) and on the level of instances (bottom). While the latter could also be achieved by means of presentation or styling, the first cannot be defined with existing RDF presentation languages.

For example, hovering is supported in Fresnel and STOOG. However, none of the languages allows for modelling further, more visualisation specific interactions such as co-highlighting graphic objects or highlighting neighboured nodes or edges of a node-link representation. More complex behaviour, such as multiple views connected by *linking and brushing* interaction techniques cannot be specified. Since visual mapping is often not modelled at all (cf. last subsection), this is not surprising. REMM and UISPIN support the definition of editing instructions with their languages, which could be seen as a kind of interaction with the generated graphic. In REMM, hints can be given how to transfer edits of the visualised data back to the source data, and UISPIN allows for the integration of SPARQL Update requests into the display definition.

Name	GSS	VPOET	Fresnel	REMM	Tal4Rdf	LeTL	OWL-PL	STOOG	UISPIN
Visual mapping (not only presentation)	(-)	-	-	-	-	-	-	-	(-)
Interaction describable	-	(-)	(-)	(-)	(-)	(-)	(-)	(x)	(-)

Table 4.18: Comparison of RDF display languages – Visual mapping and interactivity.

Language Paradigm

Only a few purely **declarative** *(LC-9)* RDF display languages exist. These include the style sheet languages GSS and STOOG Graph Stylesheets as well as Fresnel and the Fresnel-based REMM. In LeTL, only the selection of source data can be done declaratively using SPARQL, while display information cannot be stated declaratively.

The other display languages, Tal4RDF, VPOET and LeTL have an **imperative** style *(LC-10)* and describe how to turn data into a displayable document. Often code **templates** are used to merge data and formatting code *(LC-11)*. Tal4RDF, VPOET, LeTL and OWL-PL use templates that combine fragments of the target platform code, containing the display information, with the source data. All these template approaches offer branching (limited in VPOET) and

Name	GSS	VPOET	Fresnel	REMM	Tal4Rdf	LeTL	OWL-PL	STOOG	UISPIN
Declarative	x	-	x	x	-	x*	x**	x	x
Imperative	-	x	-	-	x	x	-	-	-
— Templates	-	x	-	-	x	x	x	-	-***

*) Data selection via SPARQL is declarative
**) Transformational (similar to XSLT)
***) »Templates« in UISPIN are not templates in the sense of a fragment of code used as a »mockup«, but still modelled

Table 4.19: Comparison of RDF display languages – Language paradigm.

iteration constructs to process the templates. Since templates are fragments of code in a concrete general-purpose language, the template-based approach does not allow platform-independence (cf. Sect. 4.3.2). UISPIN also uses the terms *templates*. However, here templates have a different meaning and the target platform code is not directly generated from code fragments, but represented by an RDF model. In OWL-PL, templates resemble XSLT-templates.

Platform and Visual Structure Independence

Corresponding to the two dimensions of variability we described in Sect. 4.1.2, we distinguish the language criteria platform-independence and visual structure independence, as we already did for visualisation languages. As a third criterion, we examine whether the languages support multiple visual structures and whether platform independence respectively visual structure independence and multiple visual structures do exclude each other or not. After having compared all languages by these three criteria, we discuss the platform and visual paradigm independent features of Fresnel in more detail, since the abstraction of Fresnel will be subject to further discussion in the following chapters.

Platform Independence Only a few of the languages in Table 4.20 are platform-independent, that means, display definition can be defined without referring to a specific platform *(LC-12)*. Among these are the style sheet languages GSS and STOOG Graph Stylesheets as well as Fresnel, which was designed to be reusable on multiple platforms.

In contrast, the added level of abstraction that platform-independence requires has intentionally been avoided and criticised by the designers of other languages, such as OWL-PL and VPOET, in favour of a less complicated or a more flexible approach. Hence, for the sake of simplicity, platform independence is not targeted by the template approaches and display code in the language of a concrete target platform is used instead.

UISPIN offers means of variability, but it does not aim at platform-independence. General variability can be achieved by the SPIN template mechanism that allows for defining parameterisable prototypes and, thereby, serves modularity and encapsulation. Besides that, a plug-in mechanism can be used by explicitly defining base components and insertion points. Although UISPIN was not intended[27] to be platform-independent, it is not limited to HTML and other XML-based languages such as SVG can be described. Platform independence could be achieved to a certain degree, whenever it is possible to describe some presentation information in an abstract view – avoiding platform-specific tags – and refine this view for each target language.

[27] Personal communication with Holger Knublauch by e-mail, 18.01.2011

Name	GSS	VPOET	Fresnel	REMM	Tal4Rdf	LeTL	OWL-PL	STOOG	UISPIN
Platform independence	x	-	x	-	-	-	-	x	(-)
Visual structure independence	(-)	-	x	-	-	-	-	(-)	(-)
Multiple visual structures/paradigms	(-)	(x)	-	-	x	x	(x)	-	-
Shareability supported by language	x	x	x	x	-	(-)	x	x	x
Extensibility of views	-	-	x	(x)	(x)	(x)	x	x	x
Composability of views	(x)	-	x	x	x	x	x	-	(x)

Table 4.20: Comparison of RDF display languages – Variability, shareability and composability.

Visual Structure Independence For the languages we examined, visual structure independence *(LC-13)*, i. e., the possibility to define display information independently from the visual structure, can be found whenever platform independence can be found. In the case of Fresnel, the authors explicitly state that they want the display information to be »visual paradigm independent« and give the examples of nested boxes versus node-link representation. Presentation information in GSS and STOOG is basically oriented towards the node-link paradigm, but some settings, for example style settings, can be done independently of that as will be described in more detail in the next paragraph. Finally, again, UISPIN could partly be used in a visual structure independent manner (to the same amount as described above for platform independence).

Multiple Visual Structures Multiple and arbitrary visual structures *(LC-14)* can always be described by languages that extend a general-purpose language and create arbitrary text output. However, if the output is limited to a specific language, the target language will sometimes lack the power to describe certain visual structures. For example, if the output is limited to (X)HTML, as it is the case for OWL-PL or VPOET, node-link-representations cannot reasonably be represented. LeTL and Tal4RDF can render SVG as well and, thereby, have this power. Among the declarative languages, only GSS allows for choosing between multiple explicitly described visual structures (table vs. node-link). STOOG Graph Stylesheets implicitly expects a node-link structure and Fresnel does (intentionally) not make any statements on visual structure.

The possibility to describe multiple visual structures and visual structure independence are two orthogonal criteria. On the one hand, some languages allow for the description of arbitrary visual structures, while not being visual structure independent. On the other hand, some languages do not describe any visual structure (e. g., Fresnel), but they allow for display definitions that are visual structure independent. Similarly, in those cases where platform-independence is achieved, this does usually not include a platform-independent description of visual structures. Rather a specific visual structure such as node-link (e.g, STOOG) or nested boxes (e. g., with the UISPIN HTML Vocabulary) is implicitly expected, or no explicit statements on the visual structure are made at all, as it is the case with Fresnel.

Only one language very basically allows for making statements on the visual structure and defining additional structure-independent display information at the same time: This is GSS, since, although the dominant visual structure in GSS is clearly the node-link structure, there is an exception to this – the *layout* setting allows for choosing between node-link and table structure for parts of the graphic, while there are some settings being valid in the context of both structures.

classLensDomain and instanceLensDomain	define the set of resources to which a lens applies
showProperties and hideProperties	control what properties of the selected resource are displayed, in what order
mergeProperties and alternateProperties	handle cases of properties that should be displayed together or used as fallbacks (irregularity of data)
Lenses used as sublenses	specify what lens to use to show the value of a given property (possible recursion)

Table 4.21: Overview of Fresnel's browser- and visual-paradigm-independent presentation knowledge – Selection and ordering. After Pietriga et al. [PBKL06a].

Platform and Visual Structure Independence in Fresnel

Before we continue the comparison of RDF presentation languages, let us have a closer look at how independence of platform and visual structures is achieved in Fresnel, which strongly influenced the work presented in thesis. The authors of Fresnel argue that the specification of presentations, that they call *presentation knowledge* [PBKL06a], is defined by each ontology-based viewer, editor or browser in a different manner. For this reason, although these applications share many presentation definition needs, no reuse was possible between them. Fresnel is suggested to be used as a common RDF presentation language to allow reuse and exchange between tools with different audiences, different domains, and also very different architectures. The Fresnel core vocabularies are independent of the »browser/application« and the »representation paradigm« [PBKL06a]. Pietriga et al. state that the expression of more knowledge would cause the vocabulary to loose this characteristic and suggest paradigm/application-specific extensions to Fresnel. Hence, Fresnel introduces an additional level of abstraction. As a basis for discussing, whether this abstraction is justified, we now have a close look at what is the exact part of presentation definition that Fresnel abstracts.

First, we focus on the **selection and structuring part** of Fresnel. Table 4.21 gives an overview of terms that are used for selecting and structuring RDF data, without being platform- or visual-paradigm-specific. For each *lens*, which may be seen as a view, it can be stated which properties of the data are to be shown and which are to be hidden. Also the order of these properties in the presentation can be defined. Further, Fresnel offers a means to loosely relate lenses to a set of resources by using the properties fresnel:classLensDomain and fresnel:instanceLensDomain. Displaying, sorting properties and coupling lenses to resources are independent of both the final platform and a concrete visual structure.

Merging and alternating properties is especially important due to the specific character of RDF data, which possibly comes from various sources on the web and, therefore, may be incomplete. Merging means summing up different property values under one heading. Alternating means accepting multiple properties, however, only the first available will be presented. For example, one source on the web may use a property *located-in*, while another uses *has-location*. Again, this is a feature that has nothing to do with specific rendering, but can be defined equally for all platforms.

Defining sublenses offers a general mechanism to select a special lens in the context of another lens. This allows for displaying resources depending on which role they play. For example, a person should not always be rendered in full detail, but sometimes, when the person is not the main subject of interest, a summary is sufficient. This is a very general principle that is required within many ontology-based UIs.

Second, we focus on the **formatting aspects** of Fresnel. Table 4.22 gives an overview of terms that are used for formatting RDF data, without being platform- or visual-paradigm-specific. Pietriga et al. refer to the formatting that Fresnel allows as »high-level, representation paradigm independent«. Since a lot of formatting is passed to CSS style sheets, the remaining formatting instructions are limited. Still, some instructions can be shared between platforms

propertyFormatDomain	defines the set of properties to which a format applies
classFormatDomain and instanceFormatDomain	define the set of resources to which a format applies
value	controls how a property value is rendered (text, fetched image, link)
label	used to specify a human-friendly label for properties
content* (contentBefore, contentAfter, ...)	used to specify additional content to put before, after, or in-between property values[28]

Table 4.22: Fresnel's browser- and visual-paradigm-independent presentation knowledge – Formatting. After Pietriga et al. [PBKL06a].

and visual paradigms: As with lenses, Fresnel offers a means to relate formats with resources (here classes, instances and properties are possible). A further useful term is fresnel:value that allows for defining how property values should be presented. For example, instead of displaying a URL as text, the user may want to create a navigable link, or load the image at the URL and display it directly instead. Independent of the final platform, all applications that display text need to define human-readable labels that may be constructed from multiple parts of the RDF data. For example, a common setting is to use rdfs:label, but display part of the URI, if no rdfs:label exists. For some resources, also other identifiers may be made a label, e. g., for persons, the forename and surname may be concatenated. Since these options depend on the purpose of presentation, it makes sense to define them within the presentation knowledge. In Fresnel this is done using fresnel:label. Sometimes there may be the need to display text that is not part of the RDF data. The additional content can be added with the Fresnel terms fresnel:contentBefore, fresnel:contentAfter, and fresnel:contentLast. This allows, for example, for defining once, for all platforms that property values should be separated by a comma and finalised by a dot. Similarly, also text to display in the case of a missing value can be defined.

Since the question of whether another level of abstraction can be justified also needs to be asked for our visualisation approach, we discuss the pros and cons of the Fresnel abstraction in more detail in the context of the Abstract Visual Model (Chapter 6). Having had a closer look at Fresnel with respect to platform and visual structure independence, in the following we continue the comparison of RDF presentation languages by the remaining criteria *shareability*, *extensibility*, *composability* as we all the *construction of editors* and the *familiarity to users*.

Shareability

All languages support sharing display definitions to a certain degree *(LC-15)*, which is usually realised by URIs, in contrast to the visualisation languages, where URIs are not used at all for this purpose. The benefits for shareability of using RDF and URIs also drove the development of Fresnel. Except for VPOET and STOOG Graph Stylesheets, all languages use URIs to identify the created views, templates or lenses. In LeTL, sharing view definitions is supported, but can only be done within the Less web platform and views may not be reused on other platforms.

Extensibility

All languages, except for VPOET and GSS, support some mechanism to extend existing display definitions, instead of building them from scratch *(LC-17)*. This can be done by inheritance of

[28] Strictly speaking we could argue that here, a *line-up* of graphic objects (as it is the case with text) is the required paradigm here, since otherwise no notion of before or after exists. However, since text is part of almost every graphic representation, this feature is widely applicable for various visual structures.

lenses and calls to sublenses, as in Fresnel, or by calling templates within other templates as in LeTL, OWL-PL or UISPIN.

Composability

All languages we compared, except for VPOET and STOOG Graph Stylesheets, supported at least some means of composing existing view or display definitions *(LC-18)*. For Fresnel and REMM, composability can be achieved by the sublens mechanism, and, for the other approaches, by calling named templates. The most explicit composition mechanism is provided by UISPIN, which allows for the definition of plugins with insertion points. Since also queries are stored as RDF, even they can be addressed via URI. For the two languages that also support some means of visual mapping (GSS and UISPIN), composability does not apply to the visual mapping part.

Construction of Editors

In contrast to the visualisation languages from the previous section, **open-source implementations of editors** *(LC-20)* are available for all RDF display languages, except for VPOET, REMM and UISPIN. Not for UISPIN, but at least for the main modules of SPIN, an API is available under an AGPL license. An implementation of STOOG was not available at the time of writing of this thesis. However, many of the languages do not have a **restrictive schema** *(LC-19)* that could be used for deriving editors from it. Tal4RDF uses an EBNF schema to describe valid expressions in its path selector language. UISPIN is restricted via a set of SPIN rules and constraints that are evaluated by TopBraid Composer Maestro Edition. However, for the time being, no other SPIN-aware systems exist that could use this language schema. For editing OWL-PL, since it is only XSLT-influenced and not actually XSLT, only XML-, but not XSLT-tools may be used. In REMM, lenses and profiles are used to overcome the inability of pure OWL to prescribe a schema that can be used to create an editor for the efficient creation of instances. We further discuss this in Sect. 4.4.

Familiarity to users

Only VPOET expects users to learn a completely new syntax for the purpose of display definition. All other languages try to build upon popular languages users might already know, depending on their background *(LC-21)*. Fresnel aims at semantic web developers and is built upon RDF. UISPIN counts on the familiarity of webdesigners with tags from markup languages such as HTML or XML, but also on experiences with SPARQL for more complex settings. Tal4RDF builds upon TAL, a template language that is supposed to be familiar to ZOPE users (ZOPE is a python application server). Similarly, LeTL reuses the syntax of Smarty templates that are well known to PHP developers. OWL-PL benefits from its close relation to XSLT as a widely used XML transformation language. Finally, CSS, which is also familiar to web designers, inspired GSS and STOOG. All these languages, just as the popular ones they are based on, are not intended to be used by end-users directly, but rather by web developers and web designers. For a few languages such as GSS, REMM, STOOG or LeTL, UIs for end-users exist that help with configuring display definitions.

Name	GSS	VPOET	Fresnel	REMM	Tal4Rdf	LeTL	OWL-PL	STOOG	UISPIN
Restrictive, tool-usable schema	-	-	-	-	(-)	-	(x)	-	(x)
Open-source editor	x	-	x	-	x	x	x	-	-
Familiar to users	x	-	x	x	(x)	(x)	(x)	x	x

Table 4.23: Comparison of RDF display languages – Construction of editors and familiarity to users.

	Resource display	Resource sorting	Resource formatting	Conditional branching	Looping	Resource selection	Attaching presentation inf. to resources
GSS	-	styles matching nodes or edges (arcs) define layout, stroke, fill, shape..		-	-	by URI and multiple other specific selectors	gss:matchNode, gss:matchArc

Fresnel	Fresnel Lenses		Fresnel Formats;
			hooks for CSS; accumulated styles
				-	-	Basic- (URI), FSL- and SPARQL-Selector	:instanceLensDomain;
							:classLensDomain;
							:propertyFormatDomain ..

REMM	Fresnel Lenses		Fresnel formats; no CSS	-	-	only Basic Selector (URI)	as Fresnel
VPOET	embedded »VPOET macros« used for combined display, sorting, formatting and rendering			(x)	-	by URI	implicitly
Tal4Rdf	embedded templates used for combined display, sorting, formatting and rendering			x	x	path-based TAL Expression Syntax (TALES)	implicitly
LeTL	URI or SPARQL result set	while display selection is separated, templates are used for combined sorting and formatting and rendering (LeTL inherits from the Smarty Template Language)		x	x	SPARQL; by URI	restriction to a class possible
OWL-PL	embedded OWL-PL expressions (XSLT-inspired) used for combined display, sorting, formatting and rendering			x	x	OWL-PL expressions (extended XPATH)	via extra Formatting Ontology
UISPIN	SPARQL and conditional/looping tags are embedded into (RDF-based models of) XML languages. A display model is created first and then rendered.			x	x	SPARQL; by URI	uispin:instanceView, uispin:view; resourceView
UISPIN Charts	tabular SPARQL result set (spr:ResultSet)		XHTML and URLs to display Google Charts API are generated	as UISPIN			

Table 4.24: Comparison of RDF display languages – Resource display/sorting/formatting and data selection mechanisms.

4.3.3 Additional Criteria for RDF Display Languages

In the remainder of this section, we list additional criteria only applying to RDF display languages.

Display, Sorting and Formatting of Resources Table 4.24 gives an overview of how the examined RDF presentation languages handle the questions of (1) *What resources to display?* (2) *In which order?* (Sorting) and (3) *How should resources be formatted?*

Looking at the first three columns, which represent these presentation steps, we note that none of the languages separates *all* these steps. Some languages such as Fresnel/REMM and LeTL separate *some* steps though. In Fresnel, selecting what to display and in which order is performed by the »Lenses« and separated from formatting being the »Formats« task. Also in LeTL, the choice of what to display is done in a previous step (SPARQL query) before going on with formatting. GSS does not provide means to select subgraphs for display, but the whole graph is styled unless parts of the graph are excluded using gss:display = none.

Conditional Branching and Looping Conditional branching and looping constructs can be found in all languages, except for the declarative ones. VPOET offers very limited conditional branching, though, and no looping.

Resource Selection Methods and Languages The basic means of selecting resources (properties, classes or instances) is to address them by URI. Besides this, however, most of the languages also support the description of selectors by means of a dedicated selection language. While SPARQL is frequently supported (Fresnel, LeTL, UISPIN), some languages use own expression languages, which are often path-based and inspired by XPATH (FSL used in Fresnel, TALES used in Tal4Rdf and OWL-PL expressions). Path-based languages are often considered a more concise or intuitive [Cha09] alternative to SPARQL, but may also coexist.

Attaching Display Information to Resources In the last column of Table 4.24, we compare how presentation information is »attached« to the resources. This is an interesting detail, since a similar mechanism is required for visualisation information as well.

While Fresnel and GSS select the resources in question by means of selectors that are attached to the *Styles* respectively *Lenses* and *Formats*, in UISPIN, the opposite approach was chosen. Here, a view can be attached to a resource, which may be a class or instance. For classes, it is possible to attach views in two different ways in order to distinguish styling the instances of a class from styling the class itself[29]. This is achieved by using the instanceView instead of the view property. In the template approaches no explicit coupling between classes and styles is possible, but data is referenced ad hoc within display descriptions. An exception to this is LeTL, which allows for restricting templates to special classes. A further observation is that only Fresnel allows for styling properties, i. e., making explicit statements on how relations should be styled. This is interesting, since we aim at the definition of visualisation information *per relation.*

Details of Resource Display and Formatting

Table 4.25 offers more details for those readers interested in a detailed comparison of RDF presentation languages. The criteria are again separated into display and formatting of resources:

Since RDF is a graph, **cycles** may occur and need to be handled somehow in order to obtain a document structure. Fresnel turns the graph into an ordered tree for this purpose. Since sublens relationships can cause infinite loops, this can be addressed by specifying a maximum recursion depth. In GSS, there is no need to handle cycles, since the graphs spatial structure is

[29] The problem of how to display instances having multiple types is not addressed here.

left as is and not turned into a document (tree) structure at all. Tal4Rdf restricts each node to show up only once in the presentation and thereby avoids infinite recursion problems. For the other languages, we could not find means of handling cycles explicitly. UISPIN and LeTL seem to delegate this problem to SPARQL (a *DISTINCT* SPARQL result set cannot contain the same statement multiple times).

The nature of RDF data, which could be gathered from various sources on the web, suggests additional language features that we sum up as *robustness improvements* and *property bundling.* **Robustness** aims at handling missing or redundant data. Fresnel allows for defining content to be displayed if no value is available for a property as well as a list of alternative properties. If the first property is not set for a given resource, the second best is taken and so on. Less improves robustness on the system level by dereferencing URIs and downloading supplementary linked data. However, robustness is not explicitly supported by the LeTL language.

Bundling properties goes beyond alternating properties, since the created »bundle« may have its own formatting information. Fresnel supports this with its mergeProperties construct. While not making this feature explicit, also Tal4Rdf and OWL-PL offer some support for bundling properties. In OWL-PL, the »or« operator may be used in statements for the same purpose. Tal4Rdf relies on reasoning for merging properties, however, the availability of a reasoner cannot be assumed in all situations, especially in lightweight linked-data scenarios. In the template languages and UISPIN, robustness and bundling may be partly achieved by using if-then-else-statements. While this allows for great flexibility, it may suffer from the fact that display code becomes verbose.

Specific language constructs for **modifying labels and values explicitly** can only be found in Fresnel. Here it is possible to add presentation-specific labels and thereby override default[30] labels or compensate for missing labels. Further, if there are multiple values for a property, often iterating these values only for the purpose of concatenating them by »,« is necessary.

	Cycles handled	Robustness improvements	Property bundling (explicit)	Modify labels (explicit)	Add content to values (explicit)	Define value rendering type (explicit)	CSS-support	Target language
GSS	-	-	-	-	-	(x)	-	(interpreted) interactive UI, DOT, SVG
Fresnel	x	x	x	x	x	x	x	-
REMM	-	-	-	-	-	x	-	-
VPOET	-	-	-	-	-		x	HTML, also JS, CSS
Tal4Rdf	x	-	(x)	-	-	-	x	XML, other textual formats
LeTL	x	-	-	-	-	-	x	XML, other textual formats
OWL-PL	?	-	(x)	-	-	-	x	XHTML
UISPIN	x	-	-	(x)	(x)	-	x	SVG, XHTML (extendible to other XML formats)
UISPIN Charts	as UISPIN			-	-	-	-	Google Widgets code (XHTML)
	Display			Formatting				

Table 4.25: Comparison of RDF display languages – Details of display and formatting.

[30] rdfs:label is evaluated by default.

Again, only Fresnel allows for stating this concisely by language constructs (fresnel:contentBefore, fresnel:contentLast). UISPIN offers similar support by allowing for the definition of specific UI-templates to iterate over values or present useful labels. Some of these are already available in the TUI[31] extension.

Another feature only supported by the declarative languages is the definition of **rendering type for the values of a given property**. Fresnel, but also GSS, allow for defining, whether a URL of an image should be printed as text or the image located at the URL should be displayed. Fresnel can also define that a navigable link should be created. REMM even extends these capabilities and allows for defining hints on suitable widgets to edit the property's values.

Most languages delegate style settings to **CSS** or use similar terms and accept values defined in CSS. The latter is the case for GSS, for example.

Additionally, in the same table, the **target language** is given, unless the approach is platform-independent. While GSS, Fresnel and REMM are not bound to any concrete platform, the template approaches directly produce code in the target platform's language. The code generation can be aware of the languages schema, e. g., with XML as target language (UISPIN) or XHTML (OWL-PL), but otherwise as arbitrary text (Tal4Rdf, LeTL, VPOET) loosing the ability to statically validate the templates.

4.3.4 Conclusion – What Is Still Missing?

All the languages we compared in this section are languages for *presenting* data, i. e., turning raw (graph) data into a human readable document format. However, none of them allows for the *explicit definition of visual mappings* in a simple manner. We can further summarise that many of these languages are not (fully) declarative and depend on a specific platform or a specific visual structure. Composability, where available, does not include visualisation aspects. Besides this, the lack of a formally described, restrictive schema often prevents the derivation of editors.

However, several lessons can be learned from the RDF presentation languages that have been collected and analysed here, especially from Fresnel. Some of the RDF-specific aspects, such as »resource selection«, »attachment of presentation information to resources« and »robustness in an open web« are important for an RDF-based visualisation language as well and should be considered in such a language. In Chapter 7, we discuss, whether building an extension to one of the presentation languages mentioned here is sufficient to overcome their shortcomings or whether a completely new language is required. Further, we also need to clarify, how presentation and visualisation languages can be used in conjunction.

[31] UISPIN component library. http://uispin.org/tui.html, accessed: 13.12.2015.

4.4 Model-Driven Interfaces

A model-driven UI is a user interface that is at least in part generated from models. Additionally, models may also be used at runtime to steer the behaviour of the UI. Constraints between model elements may be turned into restrictions to guide the user. Model-driven UIs may be graphic or non-graphic.

Many approaches for generating model-driven interfaces are based on technologies from the (OMG's) *Metamodelling Technological Space*; therefore, in the following, we first introduce model-driven UIs that build on the »conventional and established« software modelling technologies such as Eclipse Ecore (cf. Sect. 2.2.6). Then, in Sect. 4.4.2, we will turn to the ontology modelling world and present approaches of generating UIs that use ontologies as models. Finally, in Sect. 4.4.3, we review how both technological spaces may be used in combination.

We distinguish the terms *Metamodelling Technological Space* and *Ontology Technological Space* in line with [WE09].

4.4.1 Metamodel-Driven Interfaces

We could find three main different types of generating UIs from models and metamodels: The first category comprises graphical editors for graphical languages, such as those that can be built with the Eclipse Graphical Modelling Framework (GMF). The second category comprises textual editors that may contain graphical elements as well, but basically work on a textual representation and offer autocomplete features to suggest valid choices. An example for this category are text editors built with the EMFText framework. A third class of approaches allows to construct models via standard GUI widgets such as selection boxes, radio buttons, text fields and wizards, e. g., the Extended Editing Framework (EEF). These classes are not meant to be fully disjoint.

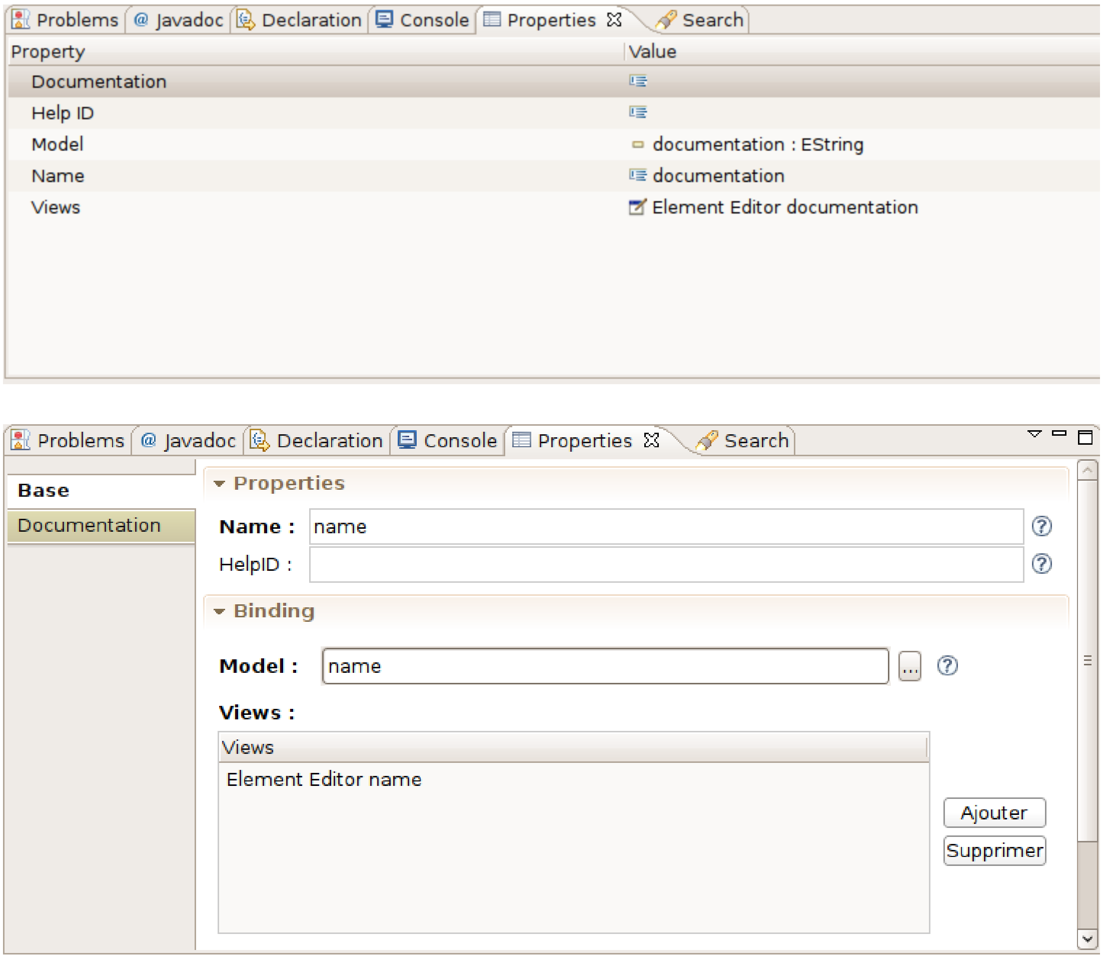
Graphical Modelling Framework

The Graphical Modelling Framework (GMF)[32] allows for building a graphical editor based on a model of the domain and an additional description of tools and graphics of the editor. All models in GMF are based on Ecore. The starting point is the Domain Model (cf. Fig. 4.15). Additionally, models describing graphical elements and tools may exist. If they do not exist already, GMF can help to derive them from the Domain Model. GMF distinguishes between Figures, which can be any graphical element, such as Line or Rectangle, and Nodes, which carry semantics such as Relation or InfoBox. After further refinement of the derived Graphical Definition and Tool Definition, domain elements are mapped to the tools and graphics. This information is stored in the Mapping Model. Finally, the GMF uses this Mapping Model to automatically create a Generator Model that describes the diagram editor, and which can then be used to generate platform-specific code.

EMFText

EMFText[33] allows for defining a text syntax for Ecore-based metamodels. Based on the schema of the language (concrete syntax and metamodel), EMFText generates editors, being built around the concrete text syntax. These editors have autocomplete features that support writing valid statements and when constraints are broken in spite of that warnings can be issued. By this, the editor allows for guidance with respect to (and driven by) both the concrete and abstract syntax definition.

[32] http://www.eclipse.org/modeling/gmp/, accessed: 02.07.2015.
[33] http://www.emftext.org, accessed: 02.07.2015.



Property	Value
Documentation	
Help ID	
Model	documentation : EString
Name	documentation
Views	Element Editor documentation

Name : name

HelpID :

Model : name

Views :

Views
Element Editor name

Figure 4.14: Extended Editing Framework (EEF) – Editor without (top) and with EFF (bottom). Images taken from http://www.eclipse.org.

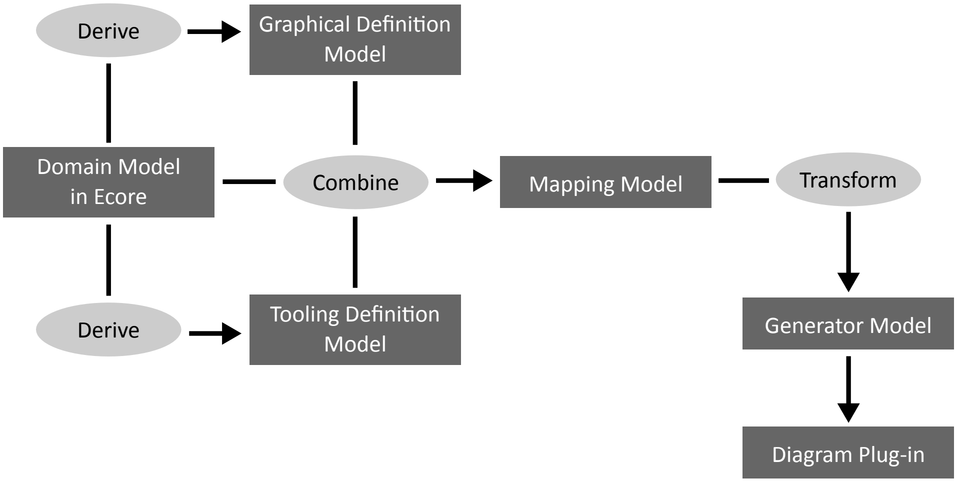


Figure 4.15: Architecture of the Graphical Modelling Framework (GMF).

Extended Editing Framework

The Extended Editing Framework (EEF)[34] extends the basic graphical Ecore editor that ships with the EMF by more complex GUI elements than the generic tree representation. Instead, it uses SWT/JFace GUI widgets such as selection boxes, sliders and wizards that can be integrated with existing GUI components (also cf. Fig. 4.14). The choice of appropriate editing widgets can partly be done automatically based on information from the metamodel (e. g., checkboxes are chosen for booleans). It can then be refined manually.

4.4.2 Ontology-Driven Interfaces

In the ontology technological space, ontology-driven (cf. Sect. 4.1.2) interfaces have been developed for several purposes already, including the convenient filtering of resources but also the editing of resources. In the following, we look at these cases in detail. A survey describing ontology-enhanced[35] UIs has been done by Paulheim and Probst [PP10]. They point out that generating user interfaces from ontologies can be seen as a special case of the MDA.

Ontology-Driven Filtering Interfaces

For filtering and exploring data, the problem of ontology-driven UIs has been successfully addressed by faceted browsing GUIs. These GUIs are always dynamically adapted at runtime, based on both the data model and a model of the users browsing state. For faceted browsing, the user does not need to have a-priory knowledge of the schema, but can start at any point and is presented all remaining options visually and instantly.

One of the first approaches to faceted browsing of RDF data was *Longwell* developed in the SIMILE project[36]. Further approaches have been compared by Delbru, Oren and Decker already in 2006 [ODD06]. They also presented the faceted browser *BrowseRDF* [DOD06], which has a high expressiveness with respect to various faceted browsing operators. An extension of the faceted browsing paradigm by weighted filtering criteria has been done by Voigt et al. [VWPM12].

Ontology-Driven Editing Interfaces

The creation and editing of (visual) mappings we are interested in, can be seen as a specialisation of the more general case of creating and editing ontological instance data. Therefore, in the following, we have a close look if and how ontology editors create UIs for creating instance data from the ontologies themselves.

OWL – Often abused for Prescriptive Constraints (Metamodelling) People who have to model ontology instance data often expect UIs to help them by presenting the options at hand and constraining them to »valid« choices. However, ontology constructs, such as domain or range and restrictions on properties are not intended to be used in a prescriptive but only descriptive manner [AZW06]. Holger Knublauch gives the following example for this common misunderstanding in a discussion of the Semantic Web community on *answers.semanticweb.com*[37]:

[34] http://www.eclipse.org/modelling/emft/?project=eef, accessed: 02.07.2015.

[35] *Ontology-enhanced* [PP10] is a less strict concept than *ontology-driven*.

[36] http://simile.mit.edu/wiki/Longwell, accessed: 02.07.2015.

[37] http://answers.semanticweb.com/questions/2329/rifspin-needed-or-not-as-when-owl2-seems-sufficient, accessed: 02.07.2015.

> » Constraint checking is particularly useful for interactive editing applications, e. g., validating user input on forms. Many people are (ab)using OWL for that purpose, e. g., to set an owl:maxCardinality constraint to »ensure« that some property can only take one value. However, doing this with OWL is usually technically incorrect, because OWL would only infer new triples. For example, assume you have maxCardinality = 1, and a resource has two values ex:Person1 and ex:Person2, then an OWL engine will tell you that ex:Person1 and ex:Person2 are the same object.
>
> *Holger Knublauch* «

Also rdfs:range and rdfs:domain are often subject to confusion as stated in other discussions[38]:

> » For similar reasons in RDFS, things like rdfs:domain and rdfs:range – which masquerade as constraints – are open to confusion. Saying that hasChild rdfs:range Person does not mean that any value for hasChild should be typed as Person: OWA [Open World Assumption] means that any value for hasChild can be automatically typed as Person, even if the data is incomplete (if x type Person is not explicitly given).
>
> *User »Signified«* «

The problem – or feature – behind these two misunderstandings described here, is due to the fact that OWL makes the Open World Assumption (OWA) and no Unique Name Assumption (UNA) [HPVH03]. Assuming an open world means that anything that is not expressed explicitly is not known, while under a closed world assumption what is not expressed explicitly is assumed to be either false or true, depending on defaults that can be stated. Assuming a unique name allows reasoning that two entities with different names are two different entities while without the UNA, as it is the case with OWL, two differently named entities may still be inferred to be the same.

Both assumptions are reasonable when inferring new data in a web of open, heterogeneous sources, where neither the completeness of data can be expected, nor can it be expected that names are always chosen uniquely. However, in some situations assuming a closed world and unique names may be beneficial, as already noted in [HPVH03]. We argue that editing data and generating input forms is such as situation, since, when editing, the user adds elements referring to his local portion of data. This is already implemented by some tools: For example, Ontology editors such as the Protégé 3.x and TopBraid Composer interpret domain–range settings as prescriptive constraints for the purpose of UI generation. Also class restrictions are evaluated (even using basic reasoning) in order to constrain the available options offered for instance editing. In TopBraid Composer, the user can override these settings by pressing a »Show all types« button.

Having said that OWL is not intended to prescribe models, nevertheless, OWL allows for defining some constraints that could be used for constraining UIs, even, when we assume the normal interpretation (with OWA and without UNA). For example, it is possible to define inconsistencies explicitly. This is a feature that can be used to check, whether the user's choices caused a contradiction. If an inconsistency occurs, the choice can then be denied.

Alternative Approaches to Modelling Constraints for Ontologies Defining prescriptive constraints that could be used to derive editors is what metamodells (cf. Sect. 2.2.6) or frames typically are used for. Interestingly, the older frame-based 3.x versions of Protégé allow for the convenient input of instances in OWL, using editing forms that are automatically generated

[38] http://answers.semanticweb.com/questions/1476/expressing-constraints-using-rdfowl-or-something-else, accessed: 02.07.2015.

from schema restrictions. This was possible, since here the OWL support was implemented as a plugin on top of an ontology system that was based on frames and used a metamodel to prescribe valid ontological models. In recent versions (starting from 4.x), Protégé does not anymore inspect the ontology schema when it builds instance data editors. This was considered inconsequent and misusage, since OWL is based on the OWA, as discussed above. Therefore, now the user is not anymore restricted when editing instances and also values not belonging to the domain of a property may now be selected. On the downside, this also means there is no more guidance for the user. This results in more possibilities as needed and selecting useful and probable values takes longer. As with textual command languages, problems arise for users that do not know the schema in advance, i. e., they do not know which values can be inserted at some given position, so they cannot search for them either.

To overcome these problems and close the gap of insufficient constraints for the description of an RDF editors various solutions have been suggested. Rauschmayer and Kiesel introduced the RDF Editing Metamodel (REMM; Sect. 4.3.1, [RK08]). Similarly, in [PENN07] »annotation profiles« are used to annotate how to configure forms for editing RDF. As in REMM, besides presentation information such as sorting and selecting parts of the ontology for editing, cardinality and range constraints can be added. The important insight here is that there are situations, where we do not want to constrain the underlying ontology, representing the world, but rather add a layer of »*complementary configuration mechanism to RDFS and OWL ontologies*« [PENN07] that applies only locally for editing purposes. A third approach to modelling constraints for RDF data is taken by TopBraid Composer based on SPIN, which we briefly introduced in Sect. 2.2.7. It can be seen as a complementary technology, designed to put (SPARQL-based) integrity constraints and rules to ontological concepts. This allows for customised closed world reasoning in addition to the use of standard (open world) reasoning for OWL. Also the general interpretation of existing OWL RL constraints under a closed world assumption can be triggered by transformation rules[39].

As an example, assume that we want to configure editing tools for a personal management ontology, and only men are working in our department. Using the aforementioned approaches, we could add a constraint that allows only men to be chosen from all available persons in order to accelerate the selection via the UI. This is in contrast to modelling such a constraint as »general truth« in the ontology. Other complementary information for editing can be hints on the preferred data types for strings or whether blank nodes should be created or not.

The approaches of REMM, annotation profiles and SPIN seem to support our idea that we need an additional means of formulating constraints, compared to what OWL offers us, when we want to define a schema for our RDF-based visualisation language in such a way that we can derive tooling from it. We come back to this issue in Chapter 7.

4.4.3 Using the Metamodelling and Ontology Technological Space in Combination

Many approaches to model-driven interfaces using technologies from the metamodelling technological space exist. However, we still want to benefit from the characteristics of the ontology technological space. Therefore, combining both spaces has to be considered [PSA+12]. Two main options can be distinguished: *Transformation* and *Bridging*. Simply transforming the ontology once into Ecore-based models and metamodels, to work with these models from then on, has limited flexibility. Changes to the underlying ontologies are not automatically available, but only after rerunning the transformation. Furthermore, it is not possible anymore to use open world reasoning on the models, once they are transformed. On the contrary, by *bridging*, we refer to approaches, where we work with the original, not a transformed ontology. This

[39] http://composing-the-semantic-web.blogspot.com/2009/01/owl-2-rl-in-sparql-using-spin.html, accessed: 02.07.2015.

second approach aims at keeping the ontology accessible in the background as long as possible (preferably also at runtime). Established (open world) reasoning services can be used on demand to infer new explicit model constraints from the ones implicitly contained in the ontology. We discuss in Sect. 8.1.3, what exactly are the benefits of this with respect to visualisation. In the following, we briefly introduce three technical approaches to bridging and comment on their applicability to our use case.

Bridging Technological Spaces with the Eclipse Ontology Definition Metamodel

The Eclipse Ontology Definition Metamodel (EODM)[40] was the first implementation of the ODM[41] using the EMF with additional parsing, inference, model transformation and editing functions. EODM was part of the IBM Integrated Ontology Development Toolkit (IODT) and used the Eclipse EMF extension to drive this integration. The toolkit enabled users to load an RDF-based ontology into EMF-based Java objects, in order to manipulate and infer the EMF Java objects representing the ontology, and to serialise the objects in RDF/XML syntax. Furthermore, it provided model transformations between RDF/OWL and other languages, such as Ecore and UML.

Unfortunately, the EODM project was terminated and is not under development anymore since 2008.

Bridging Technological Spaces with OWLText

OWLText[42] [AEWW13b] is based on EMFText, which supports defining a text syntax for Ecore-based metamodels and generating an editor for the defined DSLs. Thereby, EMFText already bridges the grammar technological space and the metamodelling technological space. OWLText additionally allows for using OWL expressions as integrity constraints in the description of a DSL schema. The OWL expressions receive their semantics as integrity constraints by being embedded into OWL Constraint Language (OWL-CL) expressions. This way, OWL-CL represents a second bridge between the metamodelling technological space and the ontology technological space.

However, OWLText can only add additional constraints to existing structures that have been modelled as (Ecore-based) metamodels before. That means, existing ontologies with their constraints are not natively supported. Since work on OWLText completed after starting our work, it has not yet been thoroughly investigated how easily OWLText may be extended into this direction. Still this may be an option for future implementations beyond the prototype that will be presented in this thesis.

Combining Technological Spaces with SPIN and TopBraid Composer

SPIN and TopBraid Composer can be seen as a further solution to combine technological spaces. However, metamodelling is here enabled by means of the SPARQL-based SPIN-constraints and SPIN-rules – not with Ecore. Standard OWL/RDFS-reasoners (ontology technological space) and SPIN-constraints and -rules (metamodelling technological space) can be applied to the same (RDF) models. This is realised by configurable chains of inference engines (Fig. 4.16). Each inference engine can offer its inferred triples as input to the next engine.

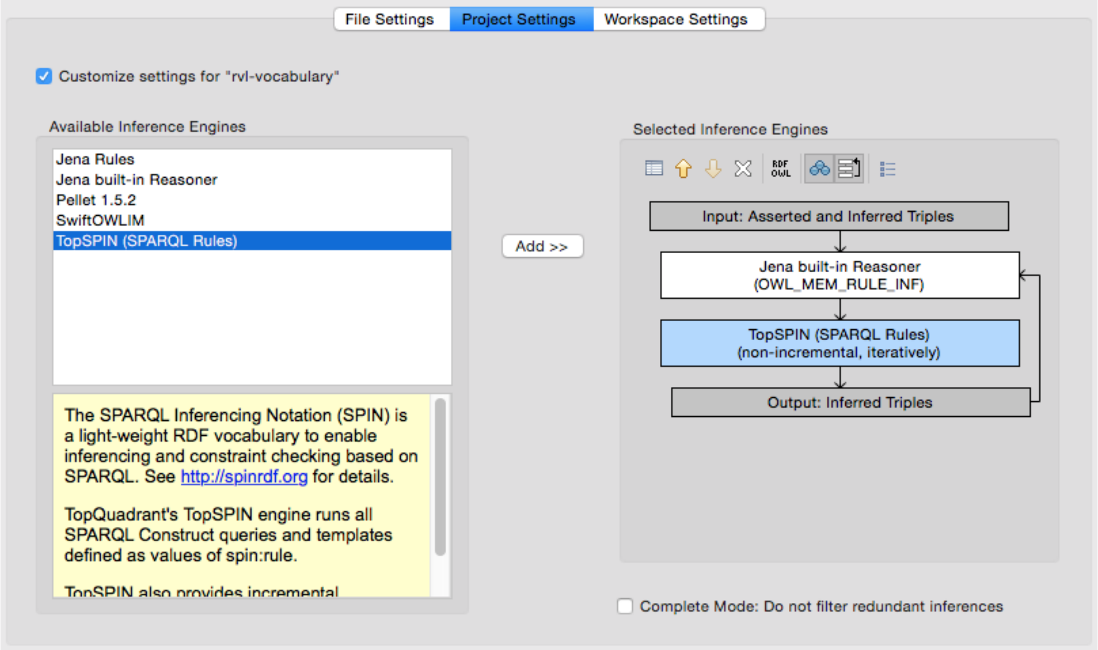
The example of SPIN – bridging the ontology and metamodelling *technologial* space – also demonstrates that we have to differentiate *technical* and *technologial* spaces: With this approach no *technical* spaces have to be bridged – both the inference of SPIN (which breaks down to SPARQL) as well as the OWL/RDFS-reasoning takes place in the same technical

[40] http://www.eclipse.org/modelling/mdt/eodm/docs/articles/EODM_Documentation/#_Toc147228127, accessed: 02.07.2015.

[41] The ontology definition metamodel (ODM)[OMG09], developed by the OMG in 2009 defines RDF(S)/OWL-based metamodels.

[42] OWLText. http://www.emftext.org/index.php/OWLText, accessed: 23.11.2015.

space of RDF. In this thesis, we use this last approach in order to have the »best from both worlds« available: Metamodelling and constraint checking with SPIN and standard RDFS/OWL reasoning (Sect. 8.1).


File Settings
Project Settings
Workspace Settings
Add >>
Input: Asserted and Inferred Triples
Jena built-in Reasoner (OWL_MEM_RULE_INF)
TopSPIN (SPARQL Rules) (non-incremental, iteratively)
Output: Inferred Triples
RDF OWL

☑ Customize settings for "rvl-vocabulary"

Available Inference Engines

Jena Rules
Jena built-in Reasoner
Pellet 1.5.2
SwiftOWLIM
TopSPIN (SPARQL Rules)

The SPARQL Inferencing Notation (SPIN) is a light-weight RDF vocabulary to enable inferencing and constraint checking based on SPARQL. See http://spinrdf.org for details.

TopQuadrant's TopSPIN engine runs all SPARQL Construct queries and templates defined as values of spin:rule.

TopSPIN also provides incremental

Selected Inference Engines

□ Complete Mode: Do not filter redundant inferences

Figure 4.16: Configuration of multiple inference engines in TopBraid Composer.

Chapter 5

A Visualisation Ontology – VISO

This chapter is based on work published in [VP11] and [PV13].

Although many terminologies, taxonomies and also a few ontologies for visualisation have been suggested, there is still no formal, accessible and reusable knowledge representation that covers the various fields of this interdisciplinary domain. For this reason, we developed a *Visualisation Ontology* (VISO) from scratch, which is applicable for our ontology-driven visualisation approach, but may be used in other visualisation systems as well. Machine-readability and interoperability is achieved using well-established Semantic Web standards such as RDF(S) [RDF04b] and OWL [OWL04].

VISO is a composite of seven ontology modules, each focusing on a different field of visualisation. Fig. 5.1 gives an overview of these modules. The graphic module formalises terms such as *Graphic attribute* and *Graphic representation*, *VISO/data* allows for characterising data variables and structures, and *VISO/activity* is concerned with the human aspects of visualisation, i. e., tasks, actions and operations. *VISO/system*, *VISO/user* and *VISO/domain* allow for describing the visualisation context and domain-specific facts. The facts module serves to formalise visualisation facts, e. g., rankings of visual means that have been described in literature, and makes this knowledge available to tools in a standardised, interoperable way.

For this thesis, the three modules graphic, data and facts will play the most import role. Fig. 5.2 shows a subset of classes, instances and properties defined in these three modules and illustrates how they are connected. The graphic module is shown in greater detail to provide an idea of how resources inside a module are linked to each other. We show the subclass hierarchy of the class *Graphic relation*[1], which divides into *Graphic attribute* and *Graphic Object-to-Object relation*. Also, concrete relations are modelled such as *Containment* as well as discrete and continuous attributes such as *shape (named)* or *saturation* (in the HSL colour model). The data module defines terms like *Scale of Measurement*, *Data Structure* and their subclasses. Finally, the facts module provides properties to relate terms from other modules according to facts found in the literature, e. g., in the rankings of expresiveness and effectiveness of graphic relations by Mackinlay [Mac86a]. Due to space limitations, only a very small part of the data and facts module can be shown in Fig. 5.2. As a concrete example of how the three modules are related, look at the graphic attribute viso-graphic:color_hsl_saturation ①. For saturation, it is stated by means of the facts module ② that it can *express* data with an *Ordinal Scale of Measurement* ③.

In the remaining sections of this chapter, we first introduce the methodology used for creating

[1] Up to this chapter, we sometimes used the less technical *visual means* instead of *graphic relation*.

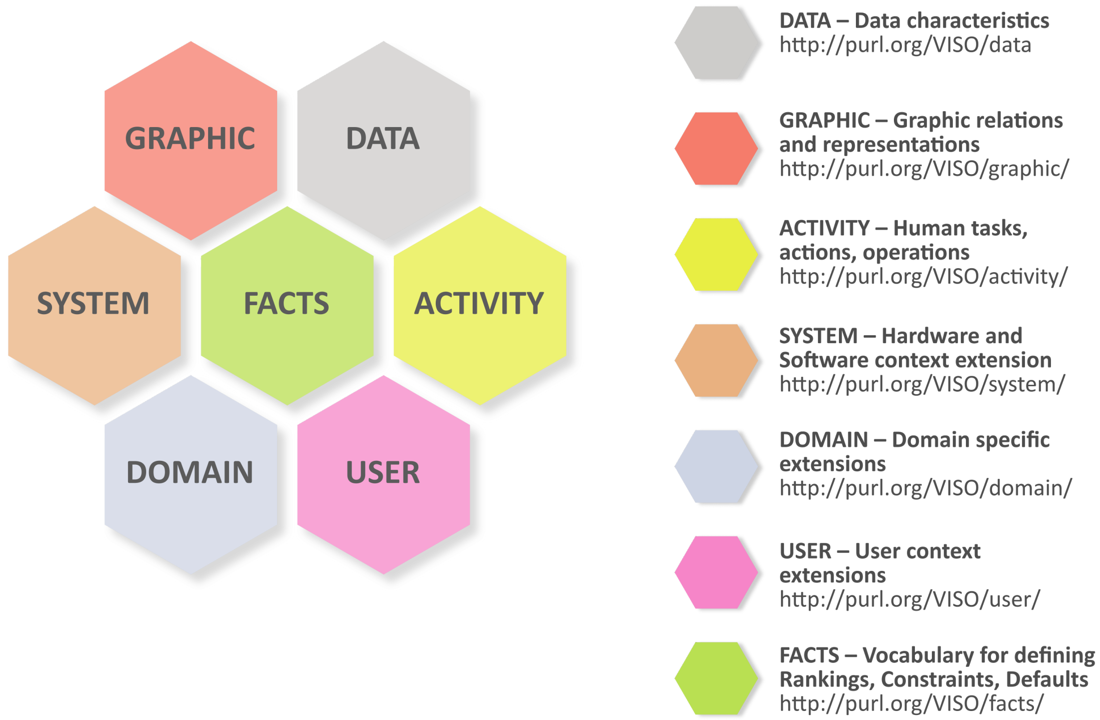

GRAPHIC
DATA
SYSTEM
FACTS
ACTIVITY
DOMAIN
USER
DATA – Data characteristics
http://purl.org/VISO/data
GRAPHIC – Graphic relations and representations
http://purl.org/VISO/graphic/
ACTIVITY – Human tasks, actions, operations
http://purl.org/VISO/activity/
SYSTEM – Hardware and Software context extension
http://purl.org/VISO/system/
DOMAIN – Domain specific extensions
http://purl.org/VISO/domain/
USER – User context extensions
http://purl.org/VISO/user/
FACTS – Vocabulary for defining Rankings, Constraints, Defaults
http://purl.org/VISO/facts/

Figure 5.1: The *Visualisation Ontology* (VISO) is a composite of seven modules, each focusing on a different field of visualisation.

the VISO ontology (Sect. 5.1) and define a set of requirements for the VISO ontology (Sect. 5.2). After this, we discuss related work on visualisation models and classifications (Sect. 5.3) and identify seven fields related to visualisation to which the existing approaches belong to.

For each of these fields, we review the existing literature, identify important terms and discuss them in detail. Thereby, we extract and align visualisation knowledge from existing work to formalise it and enter it into the VISO vocabulary. The discussion of the existing literature is done in parallel with the description of the corresponding VISO ontology module (Sect. 5.5 – 5.8).

With respect to our approach of ontology-driven visualisation, VISO serves four purposes: First, the graphic module formalises the concepts that we need to describe visual means within the declarative mapping definitions that we introduce in Chapter 7. Second, the *VISO/graphic* also precisely defines the terms and relations to be used within our Abstract Visual Model, which we introduce in Chapter 6. Third, with the data module we can describe the properties of data relations, as required to enable guidance for visualisation. Finally, using the facts and empiric-facts modules, VISO represents a source of formal, machine understandable visualisation knowledge that might be easily shared on the web.

The resulting visualisation ontology (VISO) we also consider a contribution on its own *(C-2)*, whose value is beyond the usage in our ontology-driven visualisation approach. Therefore, after briefly summarising our results in Sect. 5.9, we give an outlook on possible future work based on the ontology and point to additional usage scenarios (Sect. 5.10) and how we try to make VISO an actually popular and agreed-upon shared vocabulary (Sect. 5.11). As it was our intention, the ontology was built in cooperation and is already used in a second project.

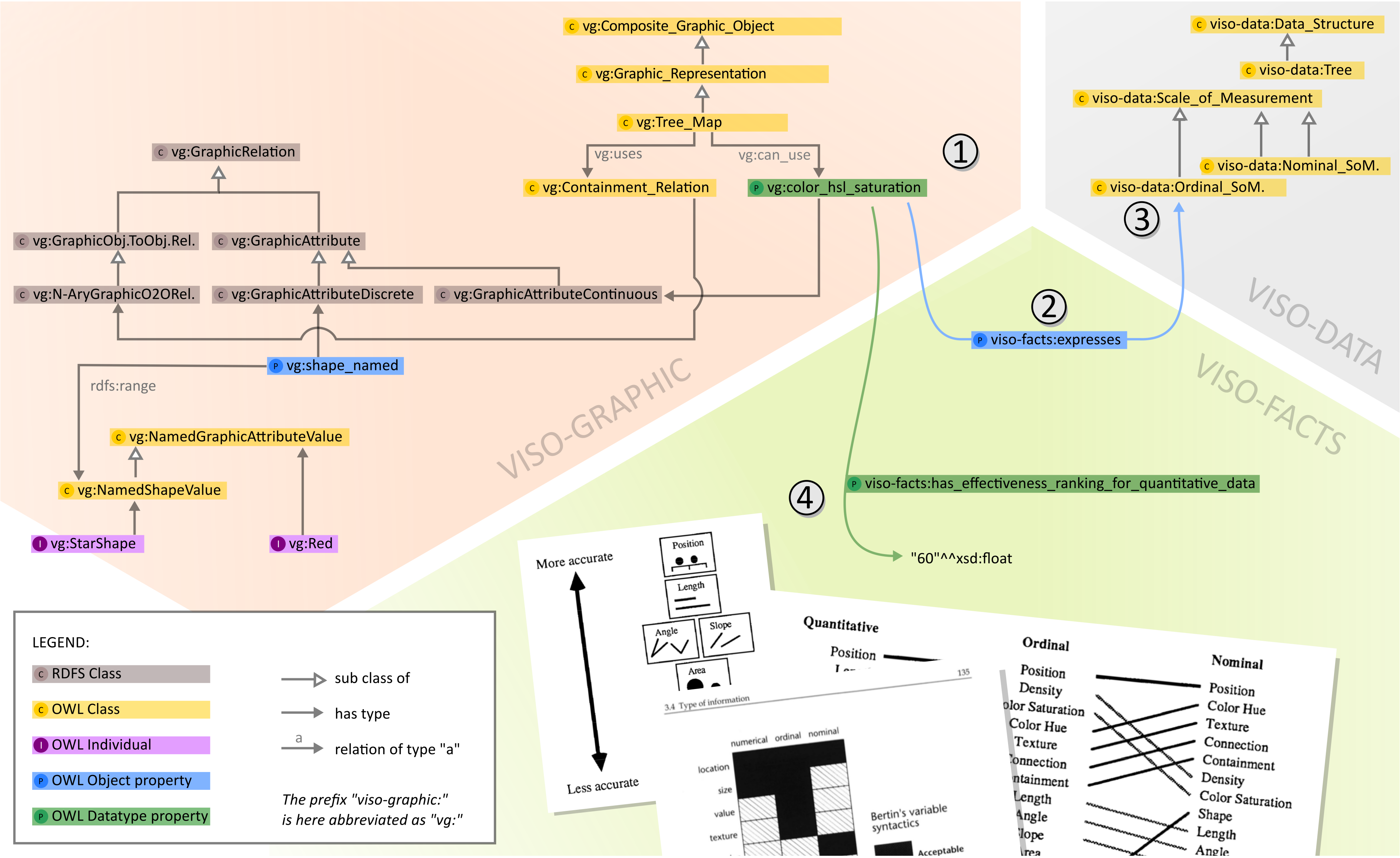


Figure 5.2: This diagram introduces some of the terms defined by the three most important modules (graphic/data/facts) and shows how they are connected using a concrete example: For the graphic attribute *Saturation* (viso-graphic:color_hsl_saturation) ①, it is stated by means of the facts module ② that saturation can express data with an *Ordinal scale of measurement* ③. Furthermore, saturation is assigned an effectiveness value for quantitative data of *»60«* (on an ordinal scale) ④. OWL and RDFS Classes are comparable to (yet not the same as) classes in object-oriented programming, while OWL individuals can roughly be thought of as instances. OWL Object and Datatype properties model relations and attributes. The images illustrating further rankings that we can formalise with the facts module are taken from [vE02, Mac86a].

5.1 Methodology Used for Ontology Creation

As a process model for creating the VISO ontology, the METHONTOLOGY [FGJ97] was chosen, which proposes to first set up requirements for the ontology, then acquire knowledge from all available sources, build semi-formal documents (glossaries, taxonomies, verb tables, concept tables, rule tables) and then do the actual formalisation. The requirements for the ontology are listed in the next section. A glossary of terms was collaboratively created using a wiki[2] and online spreadsheets. We do not show the semi-formal (intermediate) results in this document. However, for each term that was chosen to become part of the ontology, we added rich annotations pointing to the quotations from the original sources that influenced its definition (cf. Sect 5.11).

5.2 Requirements for a Visualisation Ontology

The requirements for VISO can be derived from general requirements we defined for the OGVIC approach. In the following, we precisely list the requirements and comment on their justification:

VR-1 A great variety of visual means must be defined.

VR-2 Interactive visual means must be supported (e. g., highlighting of objects).

VR-3 The visual means have to be suitable for complex, linked objects.

VR-4 Visual means need to be defined at a fine granularity to allow for flexible composition.

VR-5 Visualisation knowledge needs to be provided.

A first group of requirements concerns the content and complexity of the ontology. Being domain agnostic *(R-6)* and supporting interaction *(R-3)* requires us to formalise a large number of various visual means *(VR-1)* including interactive ones *(VR-2)*. Since we also want to have special support for ontological data *(R-4)*, we need to assure that those visual means are considered that are suitable for visualising complex linked objects *(VR-3)*. A flexible recombination of visualisation components *(R-8 and R-7)* and the ability to independently vary various visual structures used in a graphic *(R-11)*, require a fine granularity *(VR-4)* of the visual means described in the vocabulary. Finally, allowing domain experts as users of our visualisation approach *(R-12)*, requires us to provide formalised visualisation knowledge to be used by a guidance system *(VR-5)*. Examples of questions that have to be answered based on this knowledge are: »Which combinations of graphic attributes are problematic with respect to human perception?« Or: »Which visual means can I present to the user when she wants to visualise symmetric relations?«

VR-6 Visual means need to be made explicit to allow for being referenced in mapping definitions.

VR-7 The models must be formal and machine-processable.

VR-8 The models needs to be described independently of a specific platform.

VR-9 When formulating knowledge base facts, ontology terms need to be easily referenced.

[2] http://www.mediawiki.org, accessed: 02.07.2015.

A second group of requirements concerns the technical aspects of the ontology. Since we aim at using the VISO vocabulary to describe the »target side« of our explicit, declarative mapping definitions *(R-9)*, we also need to make visual means explicit *(VR-6)*. Moreover, directly using the defined visual means for a (guided) configuration of graphic representations *(R-14, R-13 and R-15)* requires us to store both the graphic vocabulary and the visualisation knowledge in a formal, machine-processable and understandable way *(VR-7)*. The criterion of platform variability *(R-10)* enforces the description of visualisation terms and knowledge to be platform-independent *(VR-8)* as well. To allow for mapping the specifics of ontologies *(R-16)*, it has to be simple to reference ontological terms within these rules *(VR-9)*.

V0-1 The models should be easily accessible, preferably by the same technologies as the data to be visualised.

Furthermore, the ontology should be easily processable by the same technologies as the data being processed and should be easily shared and integrated with existing vocabularies from the Semantic Web community (*V0-1*; optional requirement). This suggests building the ontology in the widespread and standardised RDF-based Web Ontology Language (OWL) to avoid technological gaps.

5.3 Existing Approaches to Modelling in the Field of Visualisation

Numerous models have already been developed in the field of visualisation with different goals, e. g., to classify and clarify concepts of the visualisation domain, to describe the visualisation process and even to model knowledge for visualisation systems. Already in 1986, Mackinlay started to formalise visualisation knowledge as LISP-rules [Mac86a]. Furthermore, there are multiple taxonomies on visualisation stressing different aspects of it, such as interaction, tasks or the characteristics of underlying data. In preparation of the VISO ontology, we compared a broad corpus of articles from the field of graphics and visualisation, especially those already suggesting terminologies, taxonomies and ontologies. The objective was to comprehend which fields of visualisation are covered and whether an ontology already exists that could be reused for our purposes. Further, we needed to understand what are the drawbacks of existing models and to identify important work, which could serve as a basis for the VISO ontology.

For classification models, various level of formalisation are possible. Duke et al. [DBDH05] distinguish the following three levels of formalisation for classifications: **Terminologies** introduce concepts in a less structured, informal way. **Taxonomies** define concepts in a hierarchically structured but mostly informal way. **Ontologies** are the most formal approach where concepts and their relations are based on a shared meaning. Since we require a model that can be equally processed and understood by humans and machines, a high level of formalisation is a critical factor. However, although a few initial ontologies existed in the visualisation domain, they did not cover our requirements (Sect. 5.3.2). Therefore, we extensively compare terminologies and taxonomies as well, of which many are described in the literature of the last decades (Sect. 5.3.1). Finally, we summarise other visualisation models (Sect. 5.3.3) whose main purpose is not classification.

5.3.1 Terminologies and Taxonomies

In the domain of visualisation and related areas, numerous terminologies and taxonomies have been developed with different goals, e. g., to allow for systematic reviews of existing techniques and ideas. It is not possible to discuss them all in depth in this work, hence, Fig. 5.3 illustrates the overall results of surveying 53 articles. An extensive comparison can be found in the Appendix B.1. We distinguish the field of *data* and the *domain* of the data. Further, we refer to *graphic representation* as the result of the visualisation process, which is synthesised using a *graphic vocabulary.* We subsume the topics task and interaction as *activity* (cf. Sect. 5.8). The fields *user* and *system* are about modelling the user and system context and how this benefits the visualisation process.

From our survey, we discovered three findings. In Fig. 5.3-a, one can observe that most classifications concentrate on data, graphical representation and activity. The other fields got less attention and, thus, seem to be good future directions of research. Further, only few works try to *unify* a broader spectrum of concepts of the interdisciplinary domain of visualisation, because in most cases only one or two, sometimes three areas are tackled by a single work (cf. Fig. 5.3-b). Finally, about 90% of the reviewed literature deals with terminologies or taxonomies and only 10% deals with ontologies, which we have a closer look at in the next section.

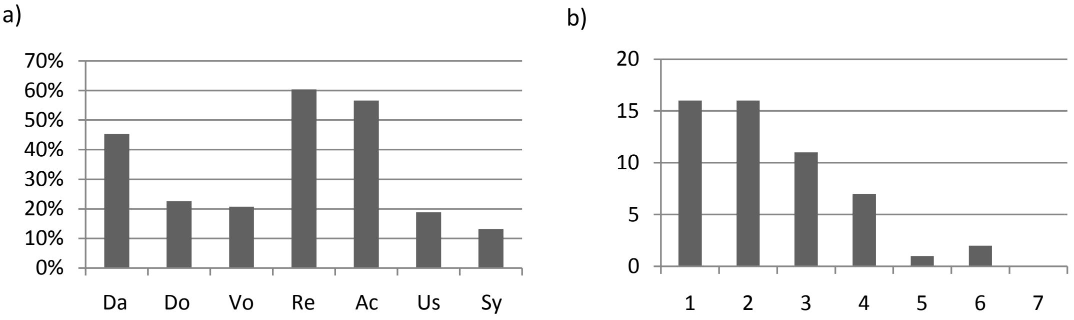


Figure 5.3: Statistical overview of reviewed literature (a) by fields they focus in general [Data (*Da*), Domain (*Do*), Graphical Vocabulary (*Vo*), Graphic Representation (*Re*), Task and Interaction (*Ta*), User (*Us*) and System (*Sy*)] and (b) by how many areas are covered within a single work.

5.3.2 Existing Visualisation Ontologies

In the following, we give an overview of existing visualisation ontologies and relate them to our objectives. Based on a workshop held at the National e-Science Centre of Scotland in 2004 [BDD^{+}04], Duke et al. advise to merge the existing visualisation knowledge fragments by means of an ontology [DBD04]. They provide a vocabulary by which users and system can communicate. It comprises only a small set of concepts and relations like data, visual representation and task. Unfortunately, an implementation or a more detailed version of their ideas are missing. Potter and Wright [PW07] took up the idea for the description of visualisation resources with focus on hard- and software requirements. Their ontology is not accessible and also misses a comprehensive overview. Rhodes et al. [RKR06] worked on an application to categorize and store information about software visualisation systems. Although they state to incorporate the concepts of Duke et al. [DBD04], the paper lacks a clear description of this fusion. Further, the developed ontology schema is tailored to software visualisation, thus, the work does not directly contribute to a domain-independent visualisation ontologies. The *Visual Representation Ontology* that Gilson et al. [GSGC08] propose as part of their tool for automatic visualisation, comprises properties of the entire graphic representation as well as attributes of single graphic objects. While this work is promising in terms of formalisation, its focus is narrow: Neither interaction or tasks nor the user are considered in this ontology. Finally, Shu et al. [SAR08] created an ontology for visualisation, intended for the semantic description of visualisation services. Based on the initial visualisation ontology [DBD04], as well as on taxonomies proposed in [BCE^{+}92] and [TM04b], their visualisation ontology mainly comprises classes for modelling data and visualisation techniques. However, they did not consider concepts like user, tasks or interaction. Furthermore, the names of numerous classes (e. g., »EnS3« or »A3_3T3«) are readable for machines but hard to understand by humans and hard to map to existing ontological concepts.

5.3.3 Other Visualisation Models and Approaches to Formalisation

The visualisation process is a complex procedure involving many steps. Therefore, several abstract models with different focuses were developed to allow for a better understanding of this process. We discussed these models already in Sect. 2.1.4. The algebras for constructing graphics as first suggested by Mackinlay [Mac86a] and later by Wilkinson in the Grammar of Graphics [Wil05] are a further source of approaches to the formalisation of graphics. However, unlike ontologies, they do not focus on the aspect of sharing knowledge and terminologies. Tory and Möller [TM04b] proposed a high-level, model-based taxonomy unifying scientific and information visualisation. Finally, the model of Brodlie and Noor [BN07] should be mentioned here, which combines a set of aspects from other existing models.

5.3.4 Summary

We distinguished seven fields the existing work on visualisation can be associated with. Most of the related work focuses only on few of these fields, while there is little work that tries to unify all of them. Further, we found that most visualisation knowledge is stored informally in terminologies and taxonomies and is not directly usable for computational reasoning.

First approaches of visualisation ontologies have emerged, but they do not sufficiently subsume and align the knowledge that is already available in existing models and classifications. Furthermore, they are not accessible to public. At the time of developing VISO, no other ontology existed that fulfilled all our requirements: None of the ontologies covered all fields of visualisation that we required to formalise for the OGVIC approach (data, graphic and the formalisation of facts), while – at the same time – being formal enough for machine processing and offering a fine granularity in the description of graphic elements.

Trying to tackle these shortcomings and build a new visualisation ontology, in the following sections, after briefly discussing technical aspects of VISO, we systematically survey the visualisation literature and discuss existing concepts and relations used by different authors.

5.4 Technical Aspects of VISO

We decided to provide the formalised graphic and visualisation knowledge in shape of an *OWL 2*[3] ontology – the *Visualisation Ontology (VISO)*. This choice was made to stay within the same technological space [BK05] as the data that is subject to visualisation. The ontology does currently not comply with the OWL-DL variant, but only with OWL-Full, due to the fact that we need to specify properties of properties (which is not generally allowed in OWL-DL).

As an effort to reflect the fields of visualisation identified in the previous section, we split the ontology into smaller parts for easier maintenance and, thereby, modularised the VISO ontology. In Fig. 5.1, we already gave an overview of the seven modules we created to partition terms and knowledge related to visualisation. We tried to create modules in a way that at least some of the modules have no dependencies to other modules. Still, most of the modules are strongly related to other modules.

In the following sections, for each module, we discuss contradicting terms, homonyms and synonyms that have been used in the literature in detail. Additionally, we present the main concepts we chose from the given field. Each term that we picked for formalisation in the

[3] OWL 2 Web Ontology Language Document Overview (Second Edition) – http://www.w3.org/TR/2012/REC-owl2-overview-20121211/

VISO is briefly introduced in a text-box summarising facts such as the name of the resource, its URI, its sub- and superclasses and a short description. A detailed documentation of the ontology terms cannot be provided here, but is available by pointing a web browser to the URI of the ontology (http://purl.org/viso/) or directly to the URIs of an ontology resource, e. g., http://purl.org/viso/Graphic_Object.

For our approach to ontology-driven visualisation, mainly the modules graphic, data and facts are relevant. Therefore, we describe these modules and the corresponding literature in more detail, while only overviewing the remaining modules. We did not constrain ourselves to these three modules, since the visualisation ontology was built to be a universal, comprehensive ontology from the beginning. It was built in cooperation with another visualisation project with a different focus, which also required to integrate the fields of human activity as well as the consideration of user and system context.

5.5 VISO/graphic Module – Graphic Vocabulary

The VISO/graphic module formalises terms used in the graphics domain such as *Graphic Relation* and *Graphic Object*. Types of *Graphic Representation* – as the final results of a visualisation process – can also be described with the vocabulary. Throughout the section, we discuss and describe terms that emerged in the context of graphical grammars and languages as suggested by Mackinlay [Mac86a], Engelhardt [vE02], Wilkinson [Wil05] and Andrienko and Andrienko [AA06]. In parallel, we present our choices of terms that we finally formalised as part of the graphics module of VISO (introduced in text boxes). Fig. 5.4 gives an overview of the most important classes of the graphic module.

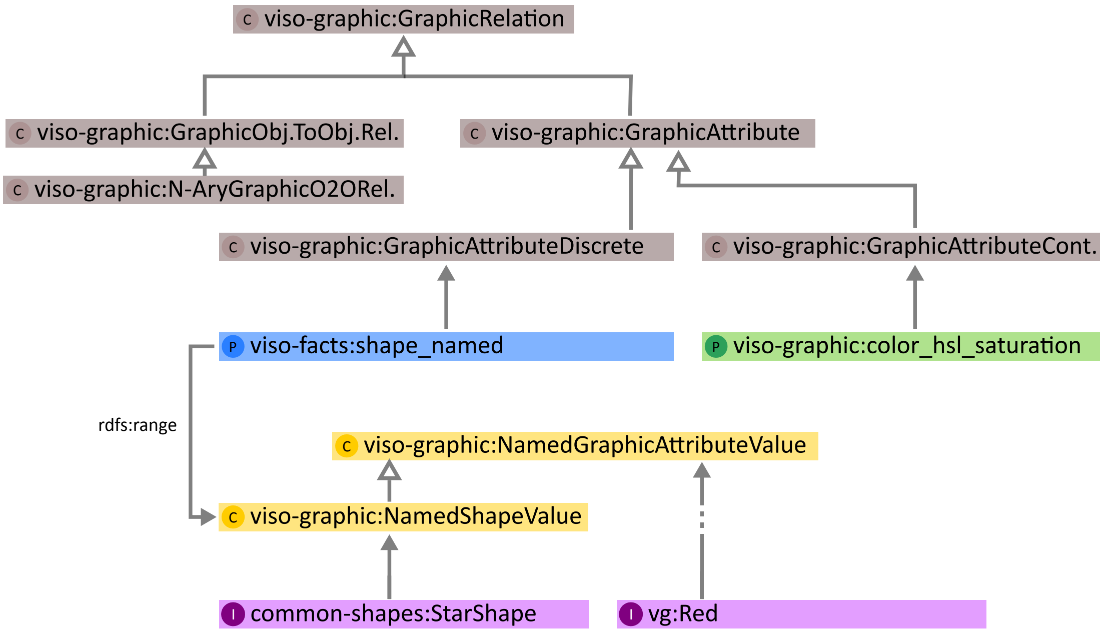

C viso-graphic:GraphicRelation
C viso-graphic:GraphicObj.ToObj.Rel.
C viso-graphic:GraphicAttribute
C viso-graphic:N-AryGraphicO2ORel.
C viso-graphic:GraphicAttributeDiscrete
C viso-graphic:GraphicAttributeCont.
P viso-facts:shape_named
P viso-graphic:color_hsl_saturation
rdfs:range
C viso-graphic:NamedGraphicAttributeValue
C viso-graphic:NamedShapeValue
I common-shapes:StarShape
I vg:Red

Figure 5.4: Main classes in the *VISO/graphic* module including examples of instances of these classes.

5.5.1 Graphic Representations and Graphic Objects

Graphic Representation
http://purl.org/viso/graphic/Graphic_Representation

A graphic representation is the result of a visualisation process consisting of graphic objects and their relations. Various subclasses allow for classifying graphic representations into named classes such as Map, Table or Node-Link diagram (primary graphic representation types) or Time Chart, Path Map (hybrid graphic representation types). Fig. 5.5 gives two examples of graphic representations currently formalised in VISO by showing concrete instances of a treemap and a time chart that have been developed in the research community. Further examples of popular graphic representations are dot plots, bar charts and pie charts.

Superclasses: viso-graphic:Graphic_Object
Subclasses: viso-graphic:Hybrid_Graphic_Representation, viso-graphic:Primary_Graphic_-Representation, ...

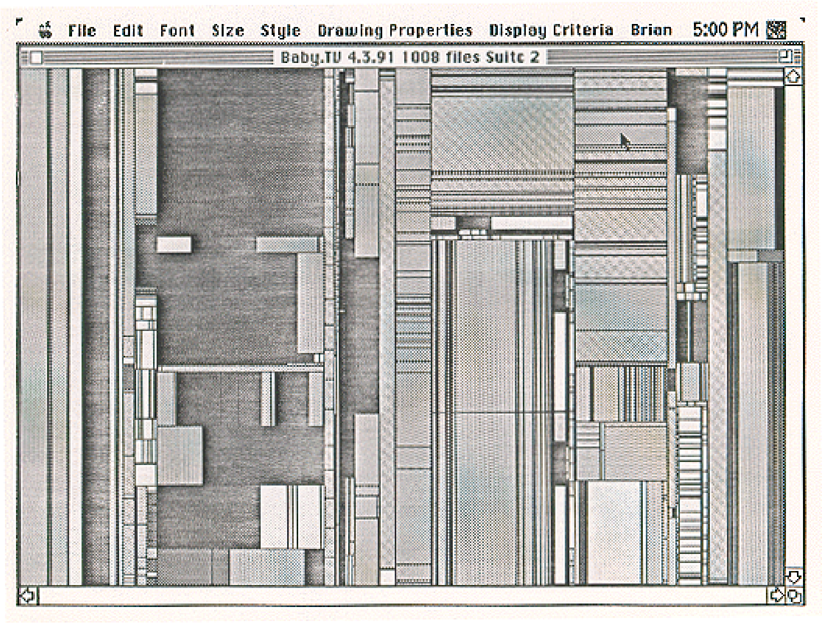
A *Graphic representation* is the result of the visualisation process. In line with Engelhardt, we distinguish *Primary graphic representations* (e. g., *Map*, *Table*, *Node-link diagram*) and *Hybrid graphic representations* (e. g., *Chronological link diagram*) [vE02]. We consider *Primary graphic representations* to be useful concepts that should appear in VISO, because users are comfortable with these terms and will watch out for common terms like *Map*. However, it is not clear if further named classes, such as those described by Engelhardt's hybrid graphic representations, make sense, since a combinatorial explosion of such classes will obviously happen for further specialisations. Also some of the *Visual Structures* listed by Card [CMS09] such as *Pie charts* or *Cone trees* can be referred to as subclasses of graphic representations according to this terminology, while he also lists techniques such as *Overview+detail* that we cannot subsume as graphic representations. Following Mackinlay [Mac86a], we permit graphic representations to be assigned a value of *expressiveness*, *effectiveness* and *appropriateness* with regard to a certain data type. We come back to these criteria in the section on facts and rankings (Sect. 5.7).

Graphic Object
http://purl.org/viso/graphic/Graphic_Object

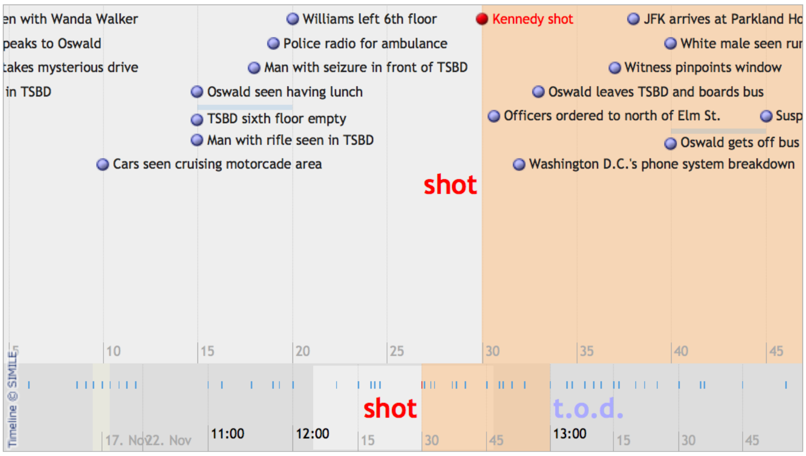
Graphic objects are the parts that make up a graphic representation. They have values for the Graphic Attributes and may be related to other graphic objects via Graphic Object-to-Object Relations. Since Graphic Representations are (composite) graphic objects as well, they can recursively be used to build more complex representations.

Subclasses: viso-graphic:Composite_Graphic_Object, viso-graphic:Elementary_Graphic_-Object, viso-graphic:Graphic_Representation

Following the composite pattern [GHJV94], a *graphic object* may be an *elementary graphic object* or a *composite graphic object*, according to Engelhardt [vE02]. A *composite graphic object* is defined as »*a graphic object that consists of a graphic space, a set of graphic objects that are contained in this graphic space, and a set of graphic relations in which these contained graphic objects are involved*«. Engelhardt further states that graphic objects carry the visual attributes such as *size*, *shape* and *colour*. He sees graphic representations as special composite graphic objects, which allows for recursion. Thereby, not only elementary graphic objects such as


File Edit Font Size Style Drawing Properties Display Criteria Brian 5:00 PM
Baby.TU 4.3.91 1008 files Suite 2

(a)



(b)

Figure 5.5: Examples of graphic representations: a) Early treemap (taken from the orginal paper on treemaps by Johnson and Shneiderman [JS91]). b) SIMILE Timeline showing events related to the death of John F. Kennedy (http://www.simile-widgets.org/timeline/, accessed: 02.07.2015).

areas or lines[4], but also composite graphic objects can carry attributes. Bertin uses the term *marks* [Ber83], however, this term seems to refer rather to the *role* that an object plays in a diagram. Therefore, we decided to use the term Graphic object in VISO.

5.5.2 Graphic Relations and Syntactic Structures

We use the term *graphic relation*[5] to refer to the things that we can vary in a visualisation including attributes such as *colour* and relative relations between graphic objects such as *containment*. Bertin used the term *visual variable* for this [Ber83]. Mackinlay puts a *basis set of primitive graphical languages* that consists of positional language categories such as *single position* and *apposed position* and the retinal variables list, but also of the categories *map*, *connection* (trees, networks) and *misc* (pie charts, Venn diagrams). However, when we consider that both attributes and complex, n-ary spatial relations can be used to visualise data, terms like *variable* or *language* seem misleading to us. *Visual variable* could also be confused with *visual attribute*. Also the terms *perceptual tasks* [CM84, Mac86a] and *visualisation primitives* [AA06] we consider problematic, since they may be mixed up with the users tasks or elementary graphic objects. We, therefore, use *graphic relation* instead.

Graphic Attribute
http://purl.org/viso/graphic/GraphicAttribute

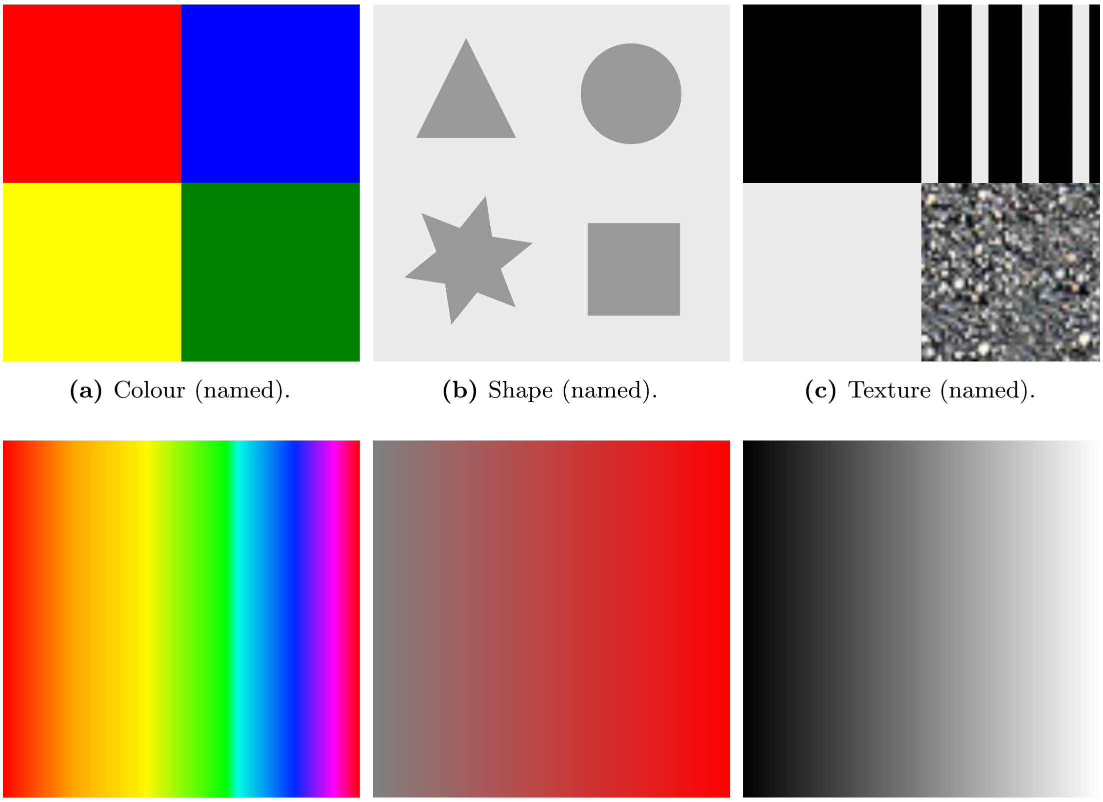
Graphic attributes represent inherent attributes of graphic objects such as *colour*, *size* and *shape*. These attributes can be distinguished from relations between graphic objects (viso-graphic:GraphicObjectToObjectRelations). In addition to the graphic attributes themselves (e. g., viso-graphic:color_named, possible named values such as viso-graphic:Red and viso-graphic:Green or viso-graphic:Bright and viso-graphic:Dark for lightness are stored in the graphic module.
Examples of graphic attributes that are currently formalised in VISO are given in Fig. 5.6.

Superclasses: viso-graphic:GraphicRelation
Subclasses: viso-graphic:GraphicAttributeContinuous, viso-graphic:GraphicAttributeDiscrete

Graphic attributes are properties of graphic objects and represent their inherent characteristics such as *colour*, *size* and *shape*. Following Engelhardt, we distinguish *graphic attributes*, which characterise a graphic attribute in a graphic space, and relations between graphic objects, which will be described below. Various variants of the term *graphic attribute* have been used in literature for similar, yet often not identical, concepts: Bertin introduced the term *retinal variables* for properties to which the retina is sensitive independent of the movement of the eye: *size*, *saturation*, *texture*, *colour*, *orientation*, *shape*. Other categories of Bertin are *positional* and *temporal variables*. Later authors reused this classification [Mac86a, Maz09]. Similarly Engelhardt classifies visual attributes into two groups: spatial attributes and area-fill attributes [vE02]. He describes the difference as follows: »*If we would regard every point of a graphic object as being anchored to its location in graphic space, then varying a spatial attribute of the object would alter this anchoring, while varying an area-fill attribute of the object would not alter this anchoring.*« Andrienko and Andrienko again use the term *retinal variables* that they see as »*internal, individual properties of marks*« and distinguish them from *dimensions*

[4] Lines – from a perception point of view – cannot actually be 1-dimensional, but must be thin areas too, in order to be visible.

[5] Up to this point, we sometimes used the less technical *visual means* instead of *graphic relation*. The remainder of this thesis will use the terms defined in VISO.



(a) Colour (named).

(b) Shape (named).

(c) Texture (named).

(d) Hue (HSL).

(e) Saturation (HSL).

(f) Lightness (HSL).

Figure 5.6: Examples of graphic attributes defined in the *VISO/graphic* module. a–c are examples of discrete »named« attributes, i. e., they will point to fixed values that are stored as resources in the ontology as NamedGraphicAttributeValue, e. g., viso-graphic:Red. Subfigures d–f show examples of continuous attributes, which point to a value that can be stored as an RDF literal.

which they describe as »*containers for marks*«. Not only spatial position, but also time plays an extra role among the visual attributes, since both are physical dimensions or spatial attributes. Although time is often less discussed, already Bertin introduced *temporal relationships* and gave *animation* as an example. Besides using time as another physical dimension, it can also be used in attributes such as *flicker frequency*.

Finally, it is important to note that some of the *graphic attributes* have multiple dimensions, or are bundled (as Wilkinson [Wil05] calls it). Position belongs to this category of attributes (three spatial dimensions) but also colour may be seen as having multiple dimensions that span a colour space (e. g., hue, saturation, lightness or red, green, blue). In *VISO/graphic*, we model each of these dimensions as a separate property, i. e., we have three properties viso-graphic:color_hsl_hue, viso-graphic:color_hsl_saturation and viso-graphic:color_hsl_lightness.

For graphic attributes that can take continuous values from a specific value range, the (maximum) value range can be stated using viso-graphic:max_value_range, which is necessary to enable default value mappings (Sect. 7.7). For example, VISO/graphic defines that viso-graphic:color_hsl_lightness can range between »0« and »100«.

Graphic Object-to-Object Relation

http://purl.org/viso/graphic/GraphicObjectToObjectRelation

The class of all properties that are visual relations between complex objects. Visual relations build visual (syntactic) object-to-object structures.
Examples of (elementary) graphic object-to-object relations that are currently formalised in VISO are given in Fig. 5.7 (a–i).

Superclasses: viso-graphic:GraphicRelation
Subclasses: viso-graphic:BinaryGraphicO2ORelation, viso-graphic:N-AryGraphicO2ORelation

viso-graphic:GraphicObjectToObjectRelation is a second subclass of viso-graphic:GraphicRelation besides viso-graphic:GraphicAttribute, cf. Fig. 5.4). To explain the origin of this concept, we now have a close look at the aforementioned »set of graphic relations« and the resulting *graphical syntactic structures*, as Engelhardt calls them [vE02]. Since he distinguishes spatial and area-fill attributes, he introduces *spatial syntactic structures* and *attribute-based syntactic structures*. He further distinguishes structures of relations in-between graphical objects and between graphical objects and graphical space. Structures involving *object-to-object relations* are, for example, *spatial clustering*, *linking*, *containment* and *superimposition*. Card calls them *topological structures* [CMS09]. An example for a structure involving object-to-space relations is a coordinate system that spans a metric space.

Andrienko and Andrienko's *dimensions* are similar to Engelhardt's spatial syntactic structures. They are concerned with everything that provides a position. This includes physical dimensions, but also *various arrangements of the display space*. Examples of arrangements are node-link-structures and discontinuous tables that resemble Engelhardt's spatial object-to-object structures.

The treatment of graphic relations and spatial structures is the field where we encountered the most diverging approaches. Engelhardt's description of object-to-object and object-to-space structures as well as the consideration of both attribute- and spatial-relationships covers all our use cases and uses coherent terms. For this reason, we picked his terms for reuse in VISO to a large extent, including the notions of object-to-object relations, which we model as viso-graphic:GraphicObjectToObjectRelation.

In Fig. 5.7, besides the elementary graphic object-to-object relations (a–i), the subfigures j–l show further relations between graphic objects, which are more complex and need to be composed from elementary relations. For example, labelling needs to be »realised« using a connector or drawing the label (relatively) close to the labelled object as shown in the figure. Also building and co-highlighting include elementary relations or attributes such as line-up, position and colour. The formalisation of the exact relation between elementary and »composite« graphic relations is subject to further research. For example, currently VISO does not allow for specifying how *labelling* should be realised. This may cause interactions with graphic relations chosen in other mappings.

5.6 VISO/data Module – Characterising Data

The goal of visualisation is the representation of *data*; therefore, data has early been considered an important aspect to classify visualisations. Most of the models and classifications from Sect. 5.3 are considering data. Several classifications of »data types« describing data sets were used in the past decades. One of the most prominent classifications in this research area is the »Task by Data Type Taxonomy« by Shneiderman [Shn96]. He distinguished between 1-, 2-, 3-dimensional, temporal, multidimensional, tree and network data; which is an abstraction from reality, because many variations of them are possible (e. g., multitrees). Keim [Kei02] suggested six data types with some overlap with Shneidermans classification but adding the categories *Text and Hypertext* and *Algorithms and Software*. Unlike Shneiderman and Keim, Heer et al. [HHC+08] are considering data from the perspective of the user. They are categorising three kinds of data: personal, community and scientific data.

Also in our approach, the characterisation of data plays an important role and is essential to allow for the effective suggestion of visual means and the correct calculation of visual mappings. Unfortunately, no mature ontology on data characteristics exists that could be reused. An initial approach that we found was described in the article »Towards an Ontology of Fields« [KV98], for which no implementation is available though. Many of the approaches we listed above suggest a simple classification of data types. We take a different approach and characterise and classify data by multiple facets, which are presented in this section.

In the following, we discuss the field of data in more detail, starting with the structure of data and its properties. Following this, we examine characteristics of data variables. The most important characteristic of a data variable in the context of this thesis is the *scale of measurement*. Additionally, for completeness, we discuss properties of data variables in the context of statistical data. Finally, we present data-related ontologies that have been integrated with VISO or could be integrated in future.

5.6.1 Data Structure and Characteristics of Relations

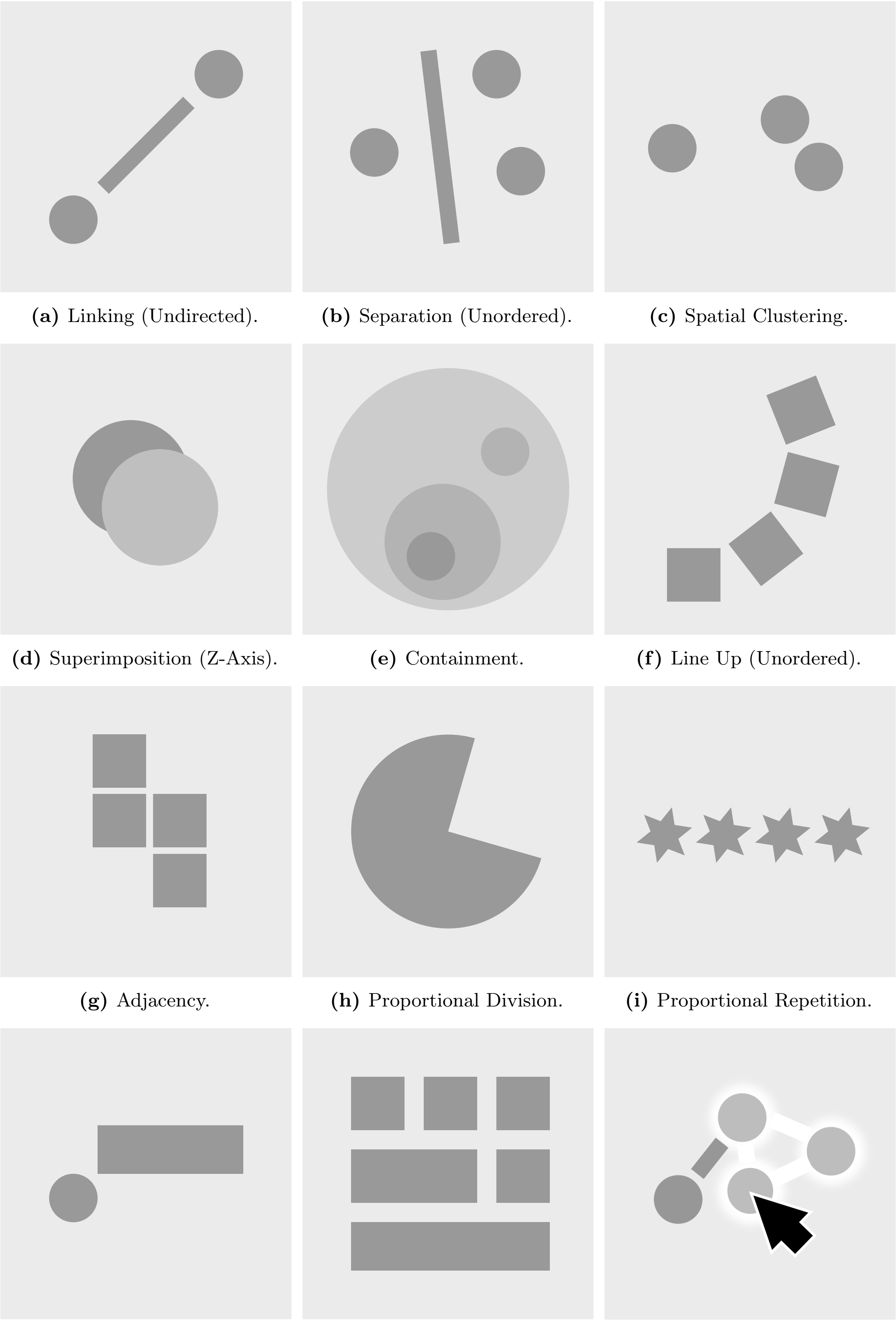
Data Structure
http://purl.org/viso/data/Data_Structure

Describes the structure that is formed by a relation. For example, the »part-of« relation will form a Directed Acyclic Graph (DAG), while the »knows« relation will only form a (general directed) Graph. Some relations will only form sequences without branching. Not every relation forms a linked structure. For some cases, e. g., »age« or »lives in« the data will only consist of a set of tuples.

Subclasses: viso-data:Graph, viso-data:DAG, viso-data:Tree, . . .

Data can be characterised by the relational structures[6], for instance, a sequence, a tree (cf. Fig. 5.8) or another possibly cyclic un-/directed graph. Besides the type of the structure, its properties including planarity, average degree of fan-in and fan-out and the existence of disconnected

[6] We refer to discrete mathematical structures in VISO, not to the corresponding structures and data models used in computing such as arrays or lists.



(a) Linking (Undirected).

(b) Separation (Unordered).

(c) Spatial Clustering.

(d) Superimposition (Z-Axis).

(e) Containment.

(f) Line Up (Unordered).

(g) Adjacency.

(h) Proportional Division.

(i) Proportional Repetition.

(j) Labelling.

(k) Building (Y-Axis).

(l) Co-Highlighting.

Figure 5.7: Overview of graphic relations. The first three rows (*a–i*) are elementary graphic-object-to-object-relations, while the last three examples represent already composed graphic relations. Also for case *i* (proportional repetition) it could be argued that it is (typically) composed with a line-up.

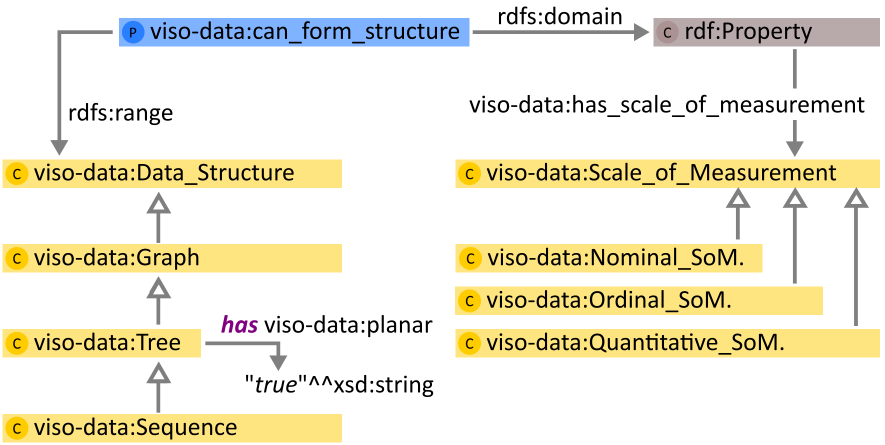


Figure 5.8: Main classes in the VISO/data module.

subsets [Spe07] can be described. Furthermore, characteristics of relations may be described such as whether the relation is reflexive, symmetric or transitive. These characteristics can already be stated in OWL. Another characteristic is the arity of a relation and whether a relation is weighted or not. Since RDFS does not allow for modelling n-ary relations directly, an *n-ary relation pattern* (cf. Sect. 6.3) has to be used. In statistical data, derived from measurements and surveys, data structure often plays a subordinated role. To provide for the high arity of the involved relations, data is often stored in a »tabular« model rather than in a graph.

5.6.2 The Scale of Measurement and Units

A (raw) data value, e. g., gained by observation, generally complies to a *scale of measurement*, which has to be available when it comes to interpreting the data [VW93]. Scales of measurement are important for visualisation purposes, since they determine the mathematical operations on the data. Similarly, if a *unit* belongs to this scale it becomes inherently connected to the values. This extrinsic information on units and scales of measurement can be stored as metadata [Wil05], which can conveniently be realised for RDF data. For tabular data, stored in spreadsheets or relational databases, often only the attribute names (the column headers) and basic data types are stored along with the raw data, while information on the scale of measurement or units has to be requested from the user during visualisation (e. g., Tableau and SPSS Viz Designer proceed like this, cf. Sect. 4.1.1).

Scale of Measurement

http://purl.org/viso/data/Scale_of_Measurement

The Scale of Measurement describes data (variables) by defining the possible mathematical operations that can be meaningfully performed on the data values. While the most frequently used scales are the *nominal*, the *ordinal* and the *quantitative* scale, other scales can be defined that refine the scales (e. g., we can refine the quantitative scale into *interval* and *ratio*) or to add domain specific criteria (e. g., a *quantitative geographic* scale).

Subclasses: viso-data:Nominal_Scale_of_Measurement, viso-data:Ordinal_Scale_of_Measurement, viso-data:Quantitative_Scale_of_Measurement, viso-data:Unstructured_Scale_of_Measurement

While we did not yet integrate ontologies for units (cf. Sect. 5.6.3), VISO can already be used to characterise the used scale of measurement, which we discuss in the following. Wilkinson [Wil05] distinguishes two possible approaches of classifying scales:

The first classification is based on Stevens' basic work on axiomatic scale theory [Ste46]. He defined four different types of scales based on the operations that are allowed on the values. These types are *nominal*, *ordinal*, *interval* and *ratio*. Combining the latter, Ware [War04] and Mazza [Maz09] are considering *categorical*, *ordinal* and *quantitative* data values. Stevens scales were detailed again by Card [CMS99] who added *spatial* (Q_s), *similarity* (Q_m), *geographic* (Q_g) and *time* (Q_t) *quantitative* scales. When calling them *time* quantitative or *spatial* quantitative, Card describes the domain of the scales of measurement. Also Engelhardt [vE02] considers *relations of physical order* and *physical distance*. The need for other than Stevens' scale types has been realised in the field of geography [Chr95] and by Marks [Mar74] who defined many more scale types. Prytulak [Pry75] criticises that every operation introduced, leads to another scale type and, consequently, the number of scale types is arbitrary. However, the general usefulness of scale types to classify variables is mostly not doubted [KS93].

As a second classification, Wilkinson differentiates between base unit classes (e. g., *length, mass, time, temperature*), secondary unit classes that are derived (e. g., *area*, *volume*, *density*) as well as dimension-less scales of measurement. Base and secondary unit classes follow *The International System of Units* (SI) [TT01].

While adopting Steven's well-known scale types, we modelled the *domain* of the data separately as an orthogonal facet. Base unit classes could be integrated from existing vocabulary (cf. Sect. 5.6.3).

5.6.3 Properties for Characterising Data Variables in Statistical Data

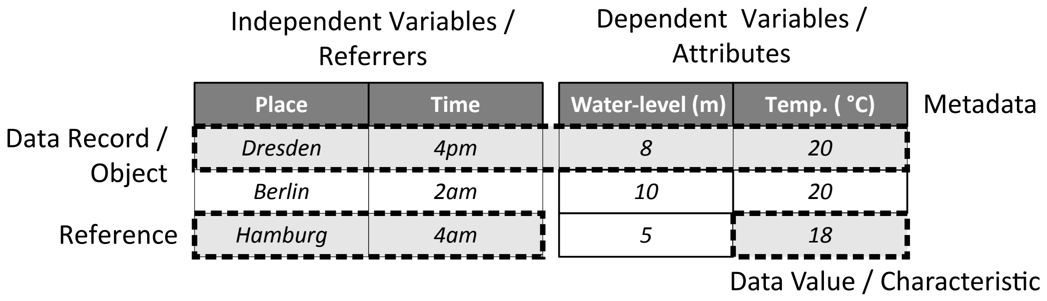
In this subsection, we compare the literature concerning various means of characterising data variables in statistical data sets. The visualisation of statistical data is not in the focus of this thesis, where we aim at supporting the visualisation of models and knowledge as represented by our case studies. Since modelling data sets and data variables is not a prerequisite for the OGVIC approach, *VISO/data* does not provide stable modelling solutions for the concepts discussed in this subsection. Furthermore, in 2014, the *RDF Data Cube Vocabulary*[7] was published, which serves exactly this purpose and is compatible with SDMX (Statistical Data and Metadata eXchange)[8], a standard for sharing statistical (meta) data. Since many existing visualisation approaches focus on statistical data and also the results of measurements and surveys may be published as Open Data in RDF, the *VISO/data* module should be aligned with the RDF Data Cube Vocabulary. This is something that we consider as future work.

In the following, we introduce basic terms for the characterisation of data variables in statistical data using the well-known tabular representation for this kind of data. Fig. 5.9 illustrates the tabular model at the example of measurement data: A *data set* (the table) is a set of *data records* / objects [AA06] (the table rows) in turns consisting of *data values* / characteristics [AA06] (the cells). It has data variables (the table columns) that we varied, the *independent variables* (to the left), and variables that we measured, the *dependent variables* (to the right; cf. [KV98]). Some authors simply use the term *object* instead of *data record* and *characteristic* instead of *data value* (e. g. [AA06]). The column headers often contain *metadata* such as units besides the name of the variable.

Independent and Dependent Variables In mathematics, variables are considered input to functions. In line with this view, Andrienko and Andrienko [AA06] see the whole data set as a function between *independent variables*, the input that is varied when we measure, and *dependent*

[7] RDF Data Cube Vocabulary, W3C Recommendation. http://www.w3.org/TR/vocab-data-cube/, accessed: 10.11.2015.

[8] Statistical Data and Metadata eXchange. https://sdmx.org/, accessed: 10.11.2015.



Independent Variables / Referrers
Dependent Variables / Attributes
Metadata
Data Record / Object
Reference
Data Value / Characteristic

Place	Time	Water-level (m)	Temp. (°C)
Dresden	4pm	8	20
Berlin	2am	10	20
Hamburg	4am	5	18

Figure 5.9: Tabular data model – Overview of terms used to describe tabular data.

variables, the output that is measured. Independent variables, are also called *referrers* [AA06] or *properties of the domain* [KV98]. The combination of all values of the independent variables is called a *reference*. Dependent variables are also called *attributes* [AA06] or *properties of the range* [KV98]. Two variables (dependent or not) may correlate, e. g., foot-size will correlate with shoe-size. Variables that have formerly been varied (as independent) can also be interpreted as dependent and the other way round [AA06]. Therefore, we suggest to model dependency as an exchangeable role of data variables in the context of a given data set.

Dimensionality Dimensionality is often seen as the number of all variables in a data set (both independent and dependent), since the term *dimension* is used synonymously with variable/attribute [Kei02, TM04b, Maz09]. Similarly, the term *multi-dimensional* is sometimes used synonymously with the term *multivariate* [YWR03]. However, for Santos [San04] dimensionality refers only to independent variables, not to all the variables – consequently he distinguishes multi-dimensional and multivariate data: Multi-dimensional data has multiple *independent* variables [San04, AA06] while multivariate data has multiple *dependent* variables [San04]. A varying meaning of dimensionality has also been noticed by Tory and Möller [TM04b]: They realised that, in their own taxonomy, *dimensionality* means »number of independent variables« when used in the context of continuous models while for discrete models it means »number of dimensions in total«. To shift around these different interpretations we speak of *dimensionality of independent variables* vs. *dimensionality of dependent variables* similar to Kemp and Vckovski [KV98].

While the dimensionality of a data set is an important criterion to describe sets of measurement data, we generally expect Linked Data sources to describe highly dimensional data. Adding more »variables« (i. e., more properties) to a selected subset of Linked Data is expected to be the normal case.

Continuous vs. Discrete Andrienko and Andrienko [AA06] distinguish continuous and discrete referrers and attributes. However, they do not state that a variable is continuous per se, but that an attribute is continuous with respect to some referrer. As a prerequisite for a continuous attribute, this referrer has to be continuous as well. An attribute is discrete, if there are only characteristics for a countable number of references possible; otherwise it is continuous. This can be aligned with Tory and Möller's [TM04b] view who understand continuity as a property of the model of data in the users mind, not as an intrinsic property of data itself. We would like to adapt this notion of continuity in future versions of VISO. One option could be to model continuity as a role of a data variable with respect to other variables, similar to the modelling of dependency.

Integration of Data-Related Ontologies

To a certain extent, existing ontologies on data have been integrated: The XML Schema Part 2 [BM04] defines primitive data types (e. g., byte, data, integer, sequence) and gives a mechanism for deriving further user-defined types. Beyond that, as early as in 1994, Gruber and Olsen [GO94] worked on the EngMath ontology, a shared notation for mathematics formalising physical quantities and dimensions. Multiple ontologies exist that are concerned with units — we only consider two recent ones from 2009 in more detail: The first is the *Quantities and Units of Measure Ontology Standard* (QUOMOS) by OASIS [OAS09], which aims at specifying the basic concepts of quantities, systems of measurement units, and scales based on the SI [TT01]. The other, the *Quantities, Units, Dimensions and Data Types in OWL and XML* Ontologies (QUDT) by TopQuadrant and NASA [MHK10] follows similar goals and is already fully specified in OWL; therefore, it could be well integrated with VISO.

5.7 VISO/facts Module – Facts for use in Visualisation Constraints and Rules

Many constraints, rules, and recommendations have been proposed in visualisation literature in order to help visualisation designers in finding not only possible, but the most appropriate visualisations for their purpose. These rules can be found in the early work of Bertin [Ber83], Tufte [Tuf83], Few [Few04], Cleveland and McGill [CM84], and Mackinlay [Mac86a]. We split the overall set of visualisation rules into four parts that we discuss in the following sections.

First, following Mackinlay's [Mac86a] definition of visualisation criteria, we distinguish statements on the *expressiveness* (Sect. 5.7.1) and *effectiveness* (Sect. 5.7.2) of graphic relations and their values. Besides these constraints, more complex rules have to be considered if we want to use multiple visual mappings in combination, as already noted in [SI94]. Therefore, we add a third section on rules concerning the composition of graphic relations and graphic representations (Sect. 5.7.3). In a last section (5.7.4), we summarise other constraints and recommendations that have been suggested by visualisation authors, e. g., with respect to the use of legends or logarithmic scaling. An additional subset of facts, which may be used in visualisation rules, is made up from collections of named values and value ranges, e. g., various sets of commonly used colours or shapes (Sect. 5.7.5).

A comprehensive collection of various visualisation rules (about 150) has been extracted from visualisation literature by Senay and Ignatius and was published as a technical report [SI90], described informally in natural language. However, for automation, a visualisation system needs visualisation knowledge as formally stated facts. This also applies to our approach to ontology-driven visualisation. Therefore, after having discussed sources of these rules, we then compare means of formalising them in Sect. 5.7.6.

When discussing the various classes of constraints and rules, like for the other VISO modules, we introduce ontology concepts and discuss related terms from literature in parallel. Although we structure this section by different kinds of rules, *VISO/facts* does only provide the vocabulary to be used in these rules as well as the vocabulary to store additional facts and rankings. It does not provide the constraints and rules themselves. Since we chose to formulate constraints and rules not with OWL, but with SPIN (Sect. 8.1.3), and because they are closely related to the visual mappings that we introduce in the next Chapter on the RVL language (Chapter 7), we deal with the actual visualisation constraints and rules separately in Sect. 8.1.

At last we introduce *VISO/facts/empiric* – an example knowledge base that we built by instantiating *VISO/facts* in order to use it in our prototypes (Sect. 5.7.7). Herein the actual facts, rankings and value collections are stored that we needed to feed our OGVIC prototypes, cf. Fig. 5.10.

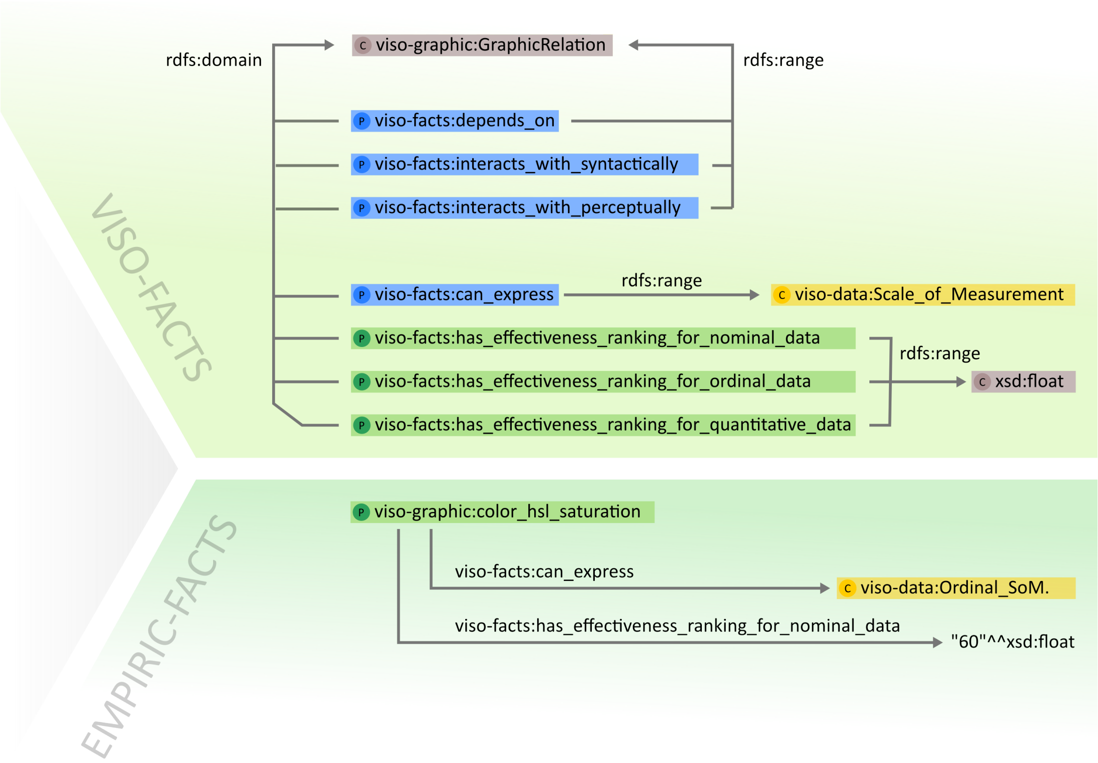

rdfs:domain
C viso-graphic:GraphicRelation
rdfs:range
P viso-facts:depends_on
P viso-facts:interacts_with_syntactically
P viso-facts:interacts_with_perceptually
VISO-FACTS
P viso-facts:can_express
rdfs:range
C viso-data:Scale_of_Measurement
P viso-facts:has_effectiveness_ranking_for_nominal_data
P viso-facts:has_effectiveness_ranking_for_ordinal_data
P viso-facts:has_effectiveness_ranking_for_quantitative_data
rdfs:range
C xsd:float
EMPIRIC-FACTS
P viso-graphic:color_hsl_saturation
viso-facts:can_express
C viso-data:Ordinal_SoM.
viso-facts:has_effectiveness_ranking_for_nominal_data
"60"^^xsd:float

Figure 5.10: Main properties in the *VISO/facts* module. Various properties from the facts module can be used to state, for example, the expressiveness and effectiveness of graphic relations. The actual statements themselves may be subject to change based on new empiric results and should be stored in extra modules. The *VISO/facts/empiric* module is one such module we provide as a starting point to store a basic set of facts (Sect. 5.7.7).

5.7.1 Expressiveness of Graphic Relations

Mackinlay sees graphic representations as »*sentences of graphical languages that have precise syntactic and semantic definitions.*« Based on this, he defines expressiveness as follows:

> » A set of facts is expressible in a language if it contains a sentence that
> (1) encodes all the facts in the set,
> (2) encodes only the facts in the set.
>
> *Jock D. Mackinlay [Mac86a]* «

That means, for a graphic relation to be expressive with respect to some data, it is not only required that all data values are covered by the graphic, but also no »incorrect« additional values may be represented, which would lead to misinterpretations. Table 5.1 shows, which of the retinal visual attributes[9] express which data with respect to its scale of measurement, according to Mackinlay. Besides clearly stating which attribute can generally express a certain

[9] Mackinlay calls them *retinal techniques*.

scale, exceptions are given. For example, Mackinlay discourages the use of size and saturation for nominal values. For colour, he indicates that not the whole colour spectrum can express order (whereas a brightness scale can do, but Mackinlay does not consider the components of colour separately). Further refinements to this table can, for example, be found in Wilkinson [Wil05], who distinguishes length, area, and volume instead of speaking about size. For area and volume, he even discourages the usage as visual attributes at all, since areas and volumes »*are not perceived in a linear relationship to the scale values*«.

can express
http://purl.org/viso/facts/can_express

States which scale of measurement the data may have that a given graphic relation can express.

Type: owl:ObjectProperty

Expressiveness of Graphic Attribute Values Statements on expressiveness are not limited to visual attributes and relations, but can also apply to sets or ranges of visual *values*. This is supported by the collection of rules in [SI90]. For example they quote rules such as the following:

- »Rule: If no one element is more important, avoid using hues of different brightness or saturations for the different [nominal] elements.« – [KPSP83] as cited in [SI90]
- »Rule: If a single qualitative variable is to be represented by symbols, the different levels of the variable should be portrayed by symbols that are graphically distinct.« – [CCKT83] as cited in [SI90]

	Nominal	**Ordinal**	**Quantitative**
Size	–	×	×
Saturation	–	×	×
Texture	×	×	
Colour	×	*	
Orientation	×		
Shape	×		

Table 5.1: »Expressiveness of retinal techniques« after Mackinlay [Mac86a] – Mackinlay discourages the use of size and saturation for encoding nominal values and constraints the usefulness of colour to encode ordinal values, since not the full colour spectrum was perceived ordinal.

Encoding Technique	**Expressiveness Criteria**
Single-position	$X \rightarrow Y$ (X is nominal)
Apposed-position	$X \times Y$ (X, Y are not nominal)
Retinal-list	$X, orX \rightarrow Y$ (X is not quantitative)
Map	$L \rightarrow X_1, \ldots$ (L is a location)
Connection	$X \times X$ (X is nominal)
Miscellaneous (single, contain, . . .)	Generally, $X \times Y$)

Table 5.2: »Expressiveness criteria for the primitive languages« after Mackinlay [Mac86a] – Mackinlay gives expressiveness constraints for a set of primitive languages that he defined. In the terminology of VISO, these comprise graphic relations such as linking (»Connection«) or colour but also compositions of those (»Apposed-position«).

5.7.2 Effectiveness Ranking of Graphic Relations

has effectiveness ranking for nominal data
http://purl.org/viso/facts/has_effectiveness_ranking_for_nominal_data

States how effective a graphic relation is ranked for nominal data.

Superproperties: viso-data:has_effectiveness_ranking_value
Type: owl:DatatypeProperty

Usually, multiple graphic relations come into consideration for expressing a certain data variable. In this case, a means for picking the best visual mapping among multiple possible ones, according to some ranking criterion, is required. Effectiveness is such a criterion, as suggested by Mackinlay. He extended the ranking of visual attributes for quantitative data by Cleveland and McGill [CM84] (Fig. 5.11) to ordinal and nominal data. Therefore, just as Cleveland and McGill's ranking, Mackinlay's effectiveness ranking is basically oriented at *accuracy* of data representation. We refer to alternative measures for effectiveness below. Furthermore, Mackinlay not only ranked attributes, but adds a few other popular visual relations such as connection and containment.

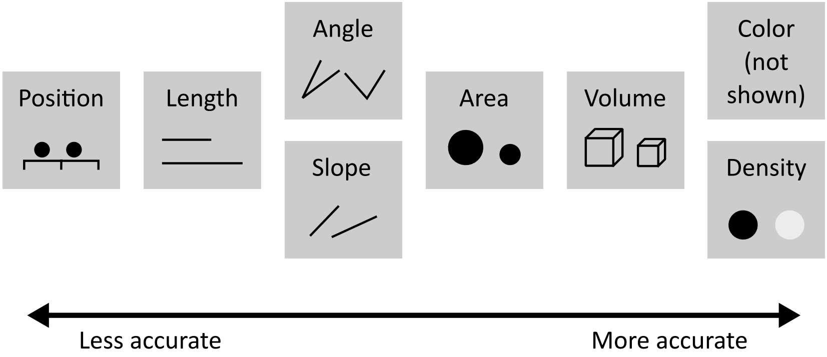


Figure 5.11: Cleveland and McGill's empirically verified ranking of quantitative perceptual tasks, redrawn as cited by Mackinlay [Mac86a] and rotated.

Looking at Fig. 5.12, especially the attribute *position* stands out, because it is rated most effective for all kinds of scales. Thereby, it becomes the most valuable attribute. All other graphic relations have a varying effectiveness depending on the scale of measurement of the data they represent. Especially *length*, *angle*, *slope*, *area*, and *volume*, which are good at encoding quantitative values, roughly change the ranking position with the three colour components (*density*, *saturation*, *hue*), *texture*, *connection* and *containment* for ordinal and nominal values. Also for ordinal and nominal values differences apply: For example, *shape* receives a high ranking value for nominal data, whereas it is rated not relevant at all for ordinal and quantitative data.

Although Mackinlay's ranking is more comprehensive, unlike Cleveland and McGill's ranking, it was not empirically verified by the author ([Mac86b] as cited in [Mac86a]). Additionally, *time*, as an additional valuable attribute for current interactive visualisations is not included.

Other Quality Measures While we adopted Mackinlay's effectiveness definition, which mainly focuses on accuracy of the graphic relation with respect to the data that needs to be represented, we are aware that this definition has been criticised [Zhu07] as incomplete, because other aspects of effectiveness exist. Zhu gives a comprehensive overview of the use of the term effectiveness in visualisation literature, pointing to other aspects such as utility and efficiency

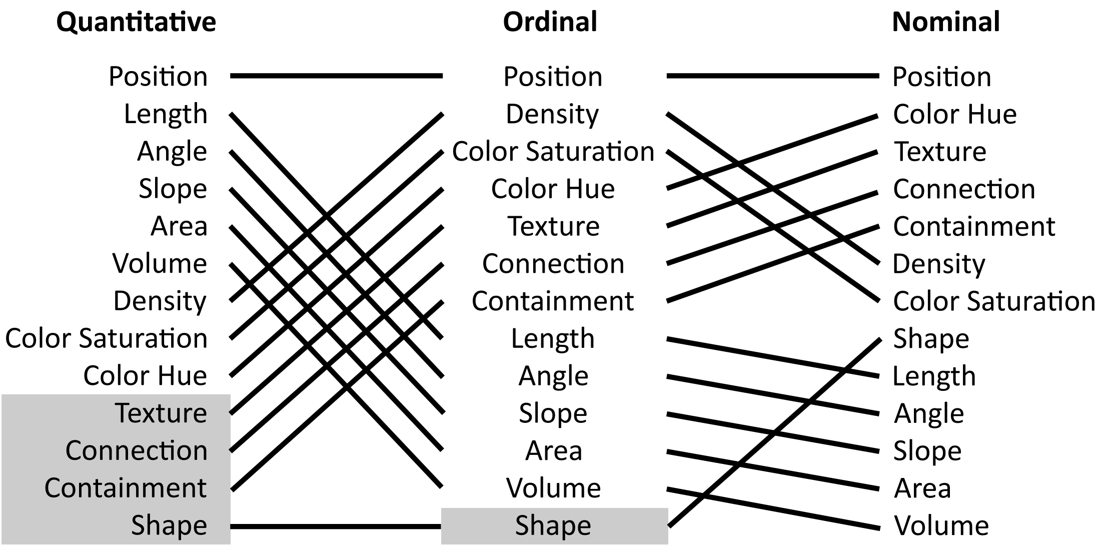


Figure 5.12: Mackinlay's extendend ranking of perceptual tasks for quantitative, nominal and ordinal values, redrawn after Mackinlay [Mac86a]. The boxed values are stated to be »not relevant« for the effectiveness ranking of the respective scale type. However, this cannot fully be aligned with the expressiveness criterion given in Table 5.1 that states, for example, that *colour* cannot express quantitative values at all.

as well. Besides contradicting definitions of effectiveness, he also points to a lack of empirical validation and means for measuring effectiveness objectively. Nevertheless, we think that our work of formalising facts and rules by using standard ontology languages can be a useful starting point. This is again supported by Zhu, who suggests that »*there needs to be a classification or taxonomy of the heuristic rules and principles*«. As a basis for this he recommends that »*a taxonomy of visualisation techniques should be developed to classify the various attributes and structures of visualization*« and that »*domain specific data classifications are needed for better accuracy assessment*«.

Effectiveness Ranking of Graphic Attribute Values Also the criterion of effectiveness can be applied to visual *values* besides visual relations. Examples from Senay and Ignatius' list [SI90] are:

- »Rule: The eye is able to perceive dark objects with greater impact than light ones.« [CCKT83] as cited in [SI90]
- »Rule: In order to be distinguishable, shapes must be quite distinct from one another. Veniar ([Ven48] as cited in [SI90]) established a differential sensitivity of 1.37 percent for shape distortion when either the horizontal or vertical sides of the square were distorted.«

5.7.3 Rules for Composing Graphics

A third class of rules concerns the composition of graphics. Senay and Ignatius, who extended Mackinlay's approach to automate the design of visualisation (APT) [Mac86a] for their visualisation system *Vista* [SI94], introduced a set of rules on composition to complement rules on expressiveness and effectiveness. These composition rules define constraints on data characteristics and apply to the five composition techniques defined by the authors. For each kind of composition, Senay and Ignatius describe rules and associated conditions that concern the »compatibility of component visualisation techniques«, the »visibility of each component upon composition« and the »distinguishability of components in the composite design«. Although, unlike expressiveness and effectiveness, the authors do not subsume composition rules as »rules of visual perception«, many rules concern the perception of the final graphic (e. g., »fields should mostly be visible through the marks of the other component«). These rules are often vague and difficult to formalise.

While we do not discuss graphic composition here in detail (we return to composition in Sect. 6.5, 6.6 and 8.2), the important thing to note is that the composition of graphics requires an additional set of facts and rules in order to respect syntactical and perceptual aspects. Based on Senay and Ignatius' composition rules as well as on statements of Wilkinson and Mackinlay on »bundling« and »interaction« of attributes, we propose the following two types of relations in which two graphic relations may take part: *Dependency* and *Interaction*.

Dependency

Dependency applies only to visual attributes and can directly be explained by the multi-dimensionality of graphic spaces [vE02]. The problem of dependent attributes needs to be considered, wherever combinations of the dimensions that span a graphic space are used for visual mapping. Two attributes are *independent*, if it is possible to set them to any value without changing the values of other variables, e. g., colour and shape can be varied independently. Visual attributes are *dependent*, if one variable is composed from the other, for example $area = height * width$. That means area and height are dependent as well as area and width. As a consequence, it is not possible to use all three involved attributes at the same time (two of them can still be used, as long as the third is not encoded). Dependency is closely related to the observations of Wilkinson, who observes that attributes can be bundled. For example, he describes *colour* as a bundle of *hue, saturation* and *brightness* [Wil05]. For independent attributes, Wilkinson also uses the term *orthogonal*. Similarly, Engelhardt calls *size* a »versatile« attribute and notes that variations of size can either be homogeneous or »restricted to height, length or width of an object« [vE02].

depends on
http://purl.org/viso/facts/depends_on

States whether a graphic attribute depends on another graphic relation, i. e., whether changing the graphic values for the one attribute will always change the values for the other attribute.

Type: owl:ObjectProperty

Interaction

By interaction we refer to a relationship between graphic relations having the potential to influence the composition of graphic relations – at least for certain values. Usually, this influence manifests itself in a negative way. Wilkinson [Wil05] remarks that »*orthogonalisation in design means making every dimension of variation that is available to one object available to another*«. That is, some graphic relations are not orthogonal or independent per se, but have to be *made* independent by putting appropriate constraints. Further, he states that »*how these variations are perceived is another matter*«. Accordingly, we subdivide interactions and the corresponding constraints into two groups: *Syntactic* interactions and constraints and *perceptual* interactions and constraints. We differentiate, whether an interaction concerns a fundamental syntactical issue – which makes it impossible to construct or decode a certain composed visualisation – or a problem with human perception. Unlike for dependency, interactions do not only apply to the composition of graphic attributes, but also to the composition of graphic object-to-object relations, as it will become clear from the examples below.

Syntactic interactions and constraints Although some visual attributes such as shape and colour are clearly orthogonal and can be varied independently, others such as shape and size are not independent per se. This is again supported by Wilkinson, who states that »*shape must vary without affecting size, rotation and other attributes*«. That means, two shapes are different and distinguishable, only if one cannot be created from the other by rotation or scaling. Hence, when both attributes shall be used, constraints have to be put on the shapes allowed for encoding (Fig. 5.13, top; the same problem arises, when no orientation exists in a graphic, e. g., at table-displays where users sit around the presentation). The second and third row of Fig. 5.13 show two examples of syntactic interactions involving object-to-object-relations: *Superimposition* can only be recognised as such, if the objects are filled, otherwise it cannot be distinguished from (overlapping) *containment*. Similarly, *transparency* can only be recognised as such, when the transparent objects do partially overlap (again *superimposition*), otherwise it cannot be distinguished from variations of colour parameters such as saturation and lightness. Syntactic constraints between object-to-object-relations also apply to *ordered line-up* and *linking*, when we try to avoid crossing connectors (Fig. 5.13, bottom). The composition with an ordered line-up makes this harder than the problem of planarity alone, because the position of nodes is not arbitrary anymore.

interacts with (syntactically)
http://purl.org/viso/facts/interacts_with_syntactically

States whether a graphic relation *syntactically* interacts with another graphic relation. Two graphic relations syntactically interact, if data encoded by these relations cannot be decoded in a unique way, leading to ambigious interpretations. We also speak of syntactic interaction, when the composition only works under additional constraints.

Superproperties: viso-facts:interacts_with
Type: owl:ObjectProperty

Perceptual interactions and constraints Perceptual interaction between two graphic relations may cause difficulties in interpreting when an encoded value takes certain values, such as extreme ones. The limitation is given by the human eye (e. g., resolution) and brain (e. g., distinction of colours). This is different from two graphic relations being in syntactic conflict, as described in the last paragraph. As an example of interaction, Mackinlay [Mac86a] describes the perceptual problem that occurs if *shape* and *size* of a visual object both encode information and *size* takes very low values (illustrated in Fig. 5.14, top row). Mackinlay already used the term interaction, however, he only refers to visual attributes, not graphic relations in general. But also the composition of object-to-object relations may be under perceptual constraints. For an example of how *lineup* and *separation by a separator* interfere with *shape* (second row) respectively *clustering* (last row), please refer to the detailed description of Fig. 5.14.

interacts with (perceptually)
http://purl.org/viso/facts/interacts_with_perceptually

States whether a graphic relation *perceptually* interacts with another graphic relation, i. e., whether the visual perception and interpretation by humans becomes difficult. In contrast to syntactic interaction, the interpretation of the composed graphic is still (theoretically) possible though.

Superproperties: viso-facts:interacts_with
Type: owl:ObjectProperty

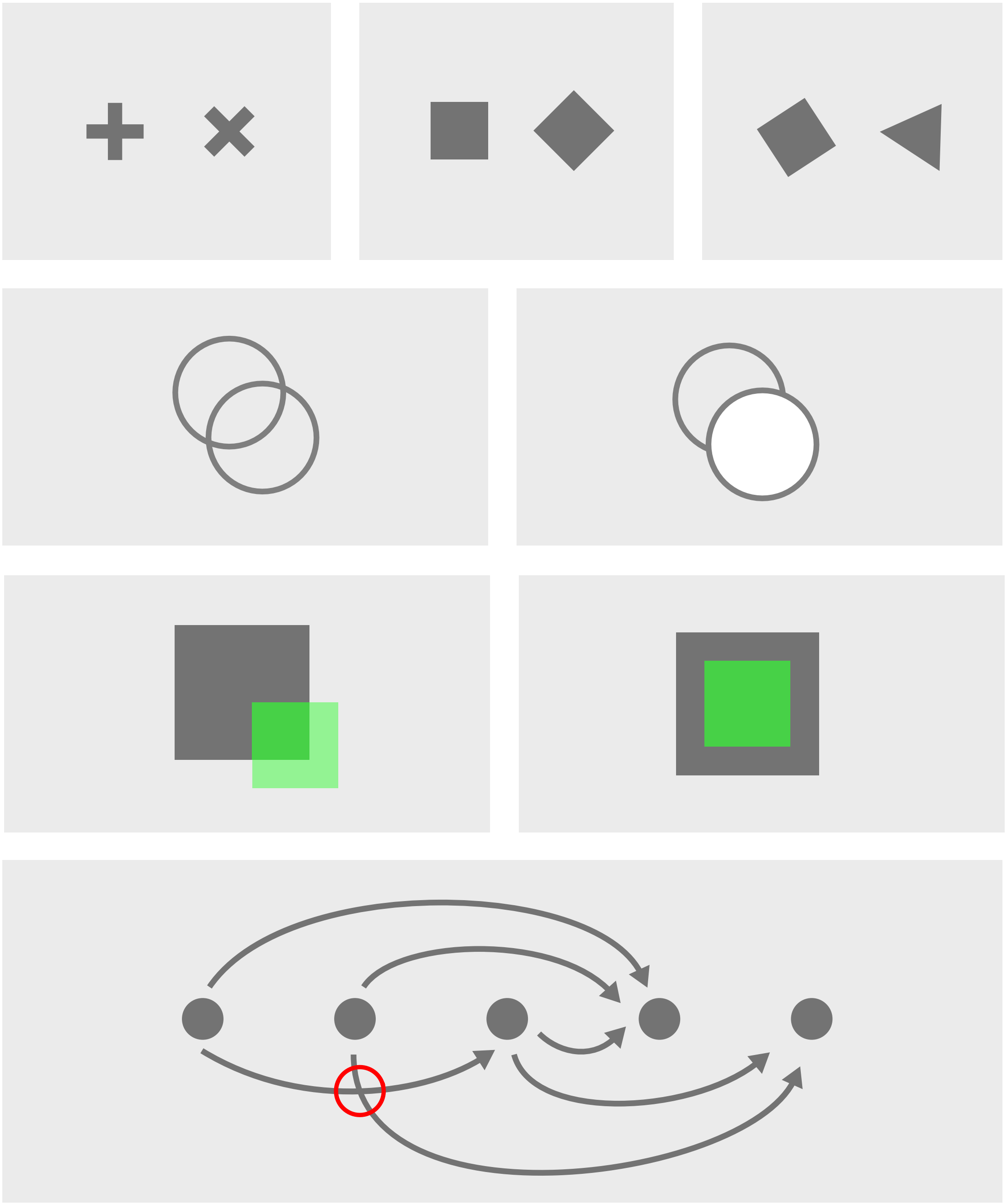


Figure 5.13: Examples of *syntactic* interaction between different graphic relations causing problems during composition. Top row: If both *shape* and *rotation* shall be used, constraints have to be put on the allowed shapes, otherwise rotating an object cannot be distinguished from changing an object's shape. Second and third row: Further examples of syntactic interactions: *Superimposition* is only perceptible if objects are filled, otherwise it could be confused with (overlapping) *containment*. Also *transparency* is only perceptible, if the transparent objects partially overlap, otherwise it could be confused with variations of colour parameters such as saturation and lightness. Bottom row: Syntactic constraints between *ordered line-up* and *linking*, when we try to avoid crossing connectors while keeping the position of the nodes.

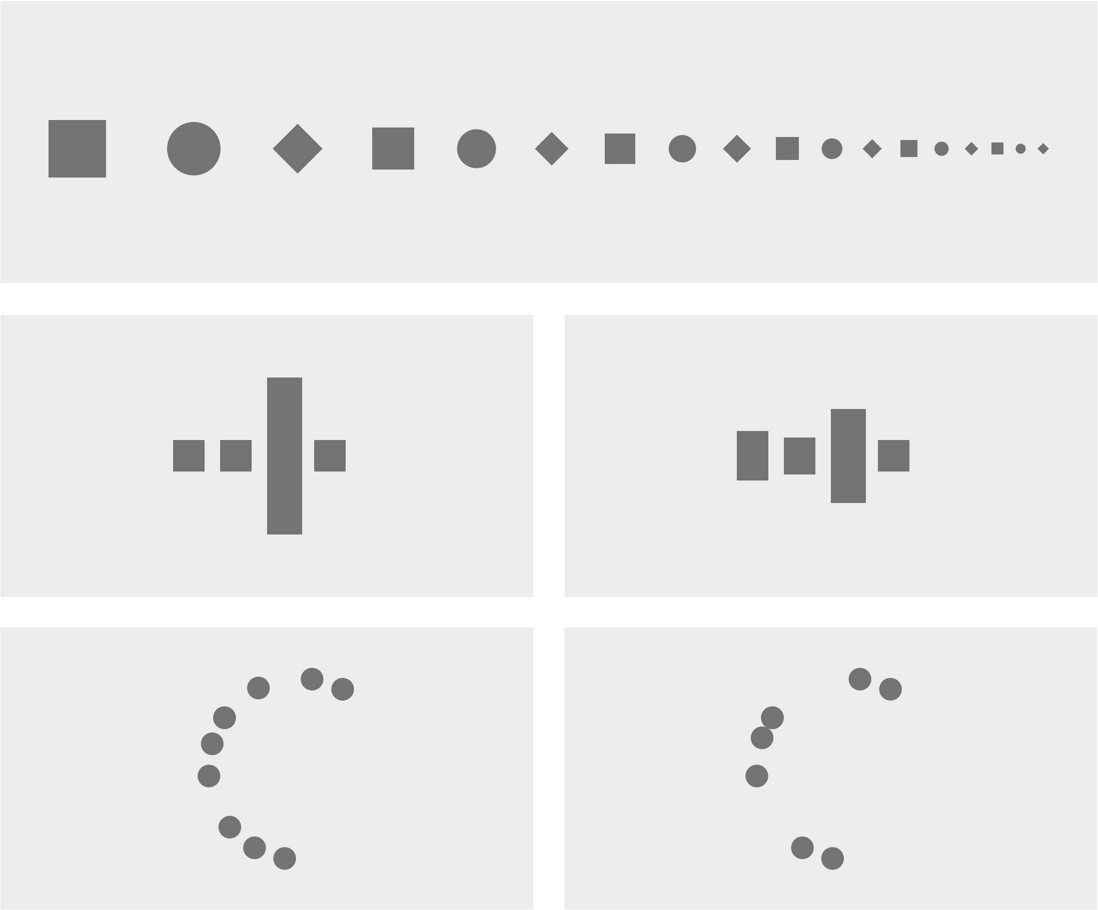


Figure 5.14: Examples of *perceptual* interaction between different graphic relations causing problems during composition. The example in the first row is taken from a figure of Mackinlay [Mac86a], titled the »Interaction between Visual Variables«, which shows how size and shape interact for small values. Other examples of such interactions are the interaction between shape and the shape used for separators, which must clearly be distinguishable from other shapes in the graphic (second row) or the interaction of clustering and lineup, since a lineup can only be perceived as such as long as the clustering does not too much distort its appearance (last row).

Constraints on the Usage of Single Graphic Relations

While, so far, we only discussed constraints for composing different graphic relations, rules exist as well for composing graphic objects by means of a single graphic relation. Again, we can differentiate between constraints that apply due to syntactic problems and those that apply due to perceptional problems. In some cases, syntactic constraints apply to this kind of compositions such as the problem of non-crossing connectors (planarity; Fig. 5.15, first row). Moreover, not every graphic relation supports every possible graph structure (e. g., containment cannot represent cyclic structures; Fig. 5.15, third row). But also perceptual constraints apply. For example, the number of graphic objects often needs to be constrained to produce a perceptually effective graphic (Fig. 5.15, bottom row).

Implementation in VISO

Currently, VISO is able to describe which graphic relations depend on each other or interact, but not *how* they do exactly. Constraints on the usage of single graphic relations cannot be specified so far.

5.7.4 Other Rules to Consider for Visual Mapping

Various other rules on visualisation exist that do not directly concern visual attributes and relations or their composition. These rules concern reference objects (e. g., the use of legends, frames, labels and background colours), scaling (e. g., when to use logarithmic scales) and issues of how to handle missing data. Many of these rules are collected in [SI90]. These rules have not yet been formalised with VISO.

5.7.5 Providing Named Value Collections

Using VISO – besides the graphic attributes themselves – named attribute *values* can be stored as individuals typed with viso-graphic:NamedGraphicAttributeValue as mentioned before in Sect. 5.5. Collections of these values can be created, e. g., by instantiating viso-graphic:Visual_Value_List. These collections represent a further part of the overall set of visualisation facts. An example for such collections are colour palettes. Special colour palettes have been defined that adhere to constraints of the human psychology or potential disabilities such as red-green blindness. An example for a tool creating such well-defined colour schemes can be found at http://colorbrewer2.org. Furthermore, Wilkinson names different scales for colour: *Rainbow Scale*, *Brightness Scale*, *Circular Scale* for circular variables, and *Bipolar scales* [Wil05]. Also for other attributes such as shape, additional facts may be formalised in order to store and share collections of common shapes and even whole symbols. For example, being able to precisely refer to and reuse values of those collections could be especially interesting, if they have been proven by a psychological test, to be well distinguishable. One example of a colour palette that we created for demo purposes (Fig. 7.10) is a visual value list containing four basic (named) colours:

```
empiric-facts:ExampleColorSetNominal a viso-graphic:Visual_Value_List ;
   rdf:first vg:Red ; rdf:rest ( viso-graphic:Green viso-graphic:Blue viso-graphic:Yellow ) .
```

Similarly, colour sets given as hexadecimal values (here a colour-blind safe set taken from ColorBrewer) can be defined:

```
empiric-facts:ColorBlindSafeColorSet a viso-graphic:Visual_Value_List ;
   rdf:first "#a6cee3" ; rdf:rest ( "#1f78b4" "#b2df8a" "#33a02c") .
```

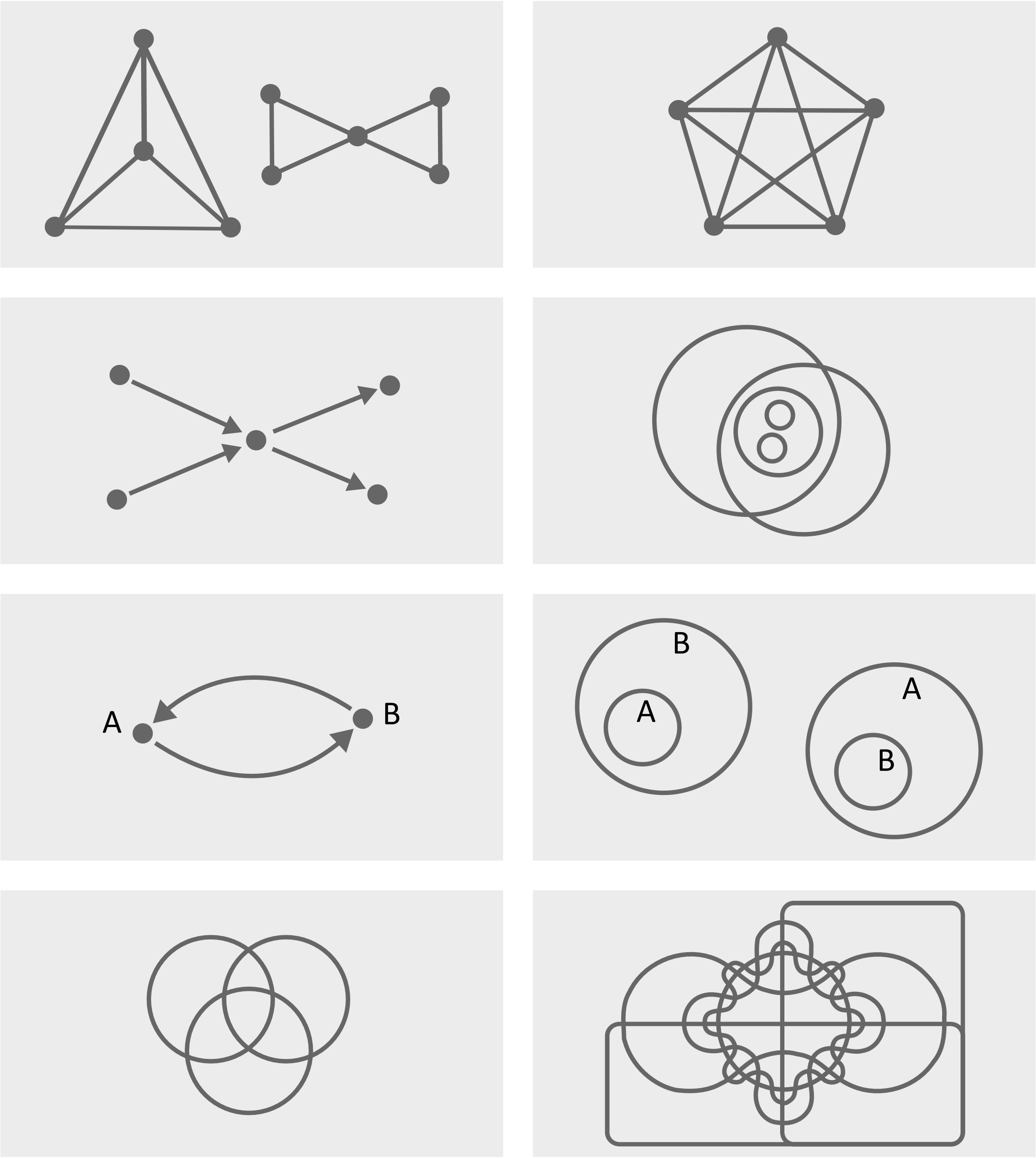


Figure 5.15: Examples of conflicts that occur if composing graphic objects by using the *same* graphic object-to-object relation multiple times. The example given in the first row shows the problem of crossing edges (non-planarity) that occurs for some node-link-diagrams. Further, not all graphic relations support all possible graph structures. For example, directed linking can represent both branching and confluent structures. Containment can represent branching as well, but representing confluence is only possible for a variant of containment (used in *Venn diagrams*) that allows for overlap (second row). Even then, the number of confluenting branches that can be visualised is limited to three for circles; higher values require to adapt the shape of the containers to ellipses and more complicated shapes. Similarly, cycles in the data are supported by directed linking, but not by containment, unless we duplicate graphic objects (third row). The last row shows that containment cannot be used arbitrary times, or will become difficult to perceive visually. While all examples involve only a single graphic relation, the first three describe a syntactic graphic problem that is impossible to overcome (at least for two-dimensional graphic representations), the last does »only« concern a limited visual perception. The graphics in the last row show a typical *Venn diagram* representing three sets (left) and a *6-set Edwards–Venn diagram* (right; redrawn from https://en.wikipedia.org/wiki/Venn_diagram, accessed: 01.12.2015). The latter technically fulfils the criterion of Venn diagrams (i. e., one container for each logical combination of sets), but is obviously much harder to perceive.

5.7.6 Existing Approaches to the Formalisation of Knowledge on Visualisation

The formalisation of rules has been done as LISP statements in Mackinlay's APT [Mac86a] and also VISTA [SI94] uses LISP-based condition-action pairs to formulate rules. Unfortunately, these rules cannot directly be used in the Semantic Web technical space, because they are not openly accessible, nor addressable on the web. Tableau and SPSS Viz Designer use visualisation rules as well, however, these rules are not stored using a common standard formalism. Expert knowledge on visualisation has also already been captured in ontologies by Gilson et al. [GSGC08]. However, they do not deal with generally applicable rules or facts on visualisation (cf. Sect. 4.1.2).

5.7.7 The VISO/facts/empiric Example Knowledge Base

As indicated above, for reasons of flexibility, the VISO module *VISO/facts*[10] only offers the vocabulary for defining facts about visualisation. It has to be complemented by an actual knowledge base, using this vocabulary. With the *VISO/facts/empiric* knowledge base we offer an example collection of facts for the purpose of this thesis and as a starting point for others. This knowledge base may be replaced in case there are new results in empiric visualisation research. The collection may also evolve into a common collection of facts if adopted by the community. The empirical facts collection contains only simple facts but no knowledge in the shape of rules or constraints. For example, the effectiveness ranking of graphic relations for specific scales of measurement is stored in *VISO/facts/empiric* but the actual constraint that uses these ranking values is stored separately in the *RVL-VISO-Constraints* collection (realised as a SPIN-constraint, cf. Sect. 8.1.3). The *VISO/facts/empiric* module may also be extended to store further common collections and ranges of attribute values as discussed in the last section.

5.8 Other VISO Modules – Activity, User, System and Domain

Since VISO was modelled to be universally usable in a broad range of use cases, additional modules have been created, which are related to the modules discussed above, but are not used in this thesis. Therefore, we only give a brief overview of these modules and point to the detailed documentation of VISO on the web[11]. These complementary modules are *VISO/activity*, *VISO/system*, *VISO/user* and *VISO/domain*.

Since visualisations are usually created for a specific purpose of the user, tasks, activities and operations can be modelled with the *VISO/activity*, following the *Activity Theory* which is well established in HCI [Nar96]. However, since our ontology-driven visualisation approach has not been extended to include these aspects so far, we do not give a detailed analysis of this field. Please refer to the technical report on VISO [VP11]. As an example for a fact that might be stored using vocabulary from the *VISO/activity* module take the following statement: »A *TreeMap supports* the *action Zoom*«. Besides view-related actions such as zooming, panning or reordering views, there are also important data-related actions such as querying, filtering or creating entities.

Further, an adaptation of visualisations based on the context, especially on a user model (*VISO/user*) and characteristics of the (hard- and software) system (*VISO/system*) could be done. Here the integration of context models, such as [PMWM08], could be taken into consideration. While such adaptations could be integrated into the ontology-driven visualisation paradigm in future, this is not within the scope of this thesis.

[10] http://purl.org/viso/facts/empiric/

[11] http://purl.org/viso/

Finally, the *VISO/domain* module bundles terms that help to relate data to a specific domain that is relevant to visualisation (e. g., geographic data, physical data). Further, also graphic representations can be marked as being frequently used in a certain domain. For example, a *Map* may be marked as being frequently used for the domain of *geography*. While the ontology-driven visualisation paradigm could be extended to consider domain-specific constraints, this is not within the scope of this thesis either.

5.9 Conclusions and Future Work

In this chapter, we described the VISO ontology, which we use for multiple purposes within the OGVIC approach: First, we use it as a target model in our visual mappings, second to formalise the concepts that we build our graphic models on, third to describe the data that is subject to visualisation, and finally as a means to share formal visualisation knowledge. With the VISO ontology, we answered the research questions *Q-1.2* and *Q-2.2* (cf. Sect. 3.2), asking for a target model to use in visual mappings and a means to formalise visualisation knowledge.

The creation of the VISO ontology became necessary, since the existing approaches to formalisation were often not accessible or did not sufficiently cover the required concepts at the right granularity. In addition to presenting the main concepts of VISO, we compared, discussed, and aligned related terms that we extracted during an extensive, systematic survey of visualisation literature, including many existing taxonomies and terminologies. We found that only a few, mostly basic, entities, e. g., *colour*, had the same meaning in all cases. In contrast, many terms had multiple meanings and also their relationships were not always in-line. An example is the dichotomy of *continuous* and *discrete*. Although many authors see this difference, the involved concepts are often slightly different – while Andrienko and Andrienko speak of continuous vs. discrete attributes, referrers and phenomena (as discussed in Sect. 5.6.3), Wilkinson [Wil05] differentiates continuous vs. categorical variables and scales.

Future work includes the integration of existing ontologies on data (RDF Data Cube Vocabulary, unit ontologies) as well as ontologies on the user and system context. Also the human activity module of the ontology should be refined to allow for more complex and adaptive applications. From the technical point of view, additional modules could be introduced, in order to achieve a reduced reasoning complexity for parts of the ontology (e. g., conformance to the OWL-DL profile) and allow for automatic consistency checking.

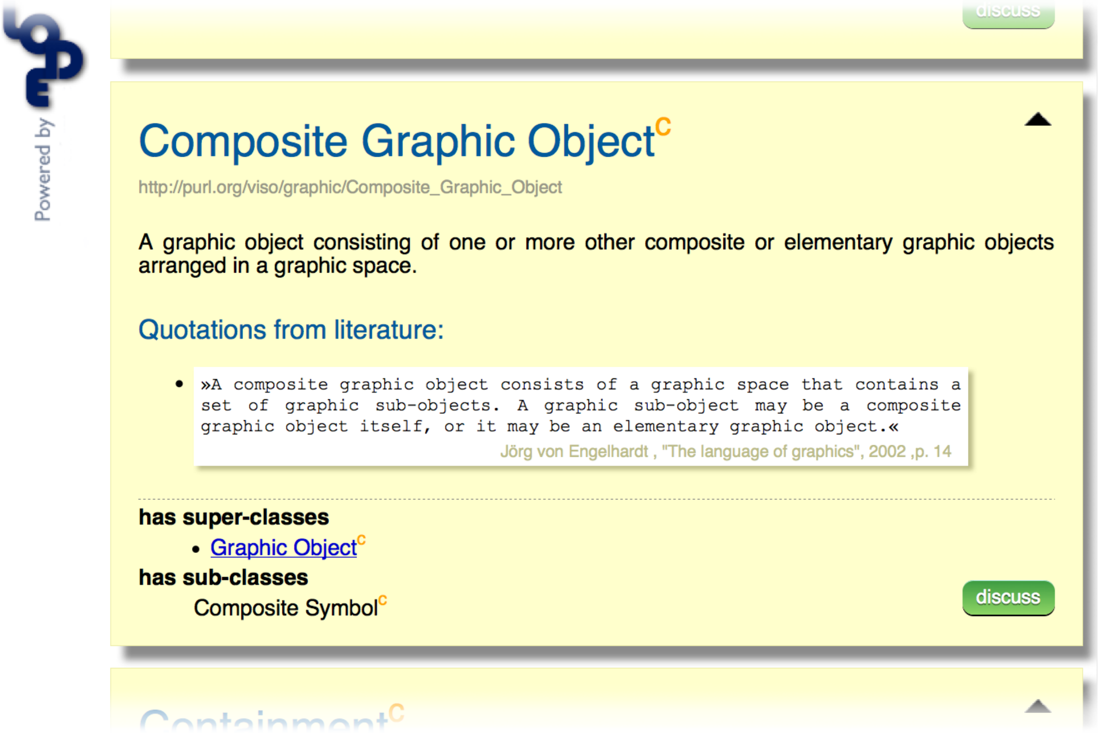
5.10 Further Use Cases for VISO

While our development on VISO was foremost motivated by the need for a widely accepted model of graphic terms and knowledge to be directly used in our approach, VISO is intended to be used in other upcoming projects as well. When building their own visualisation systems, developers have different options to employ VISO. A simple but effective example is linking to VISO resources by their URI, e. g., http://purl.org/viso/graphic/Graphic_Representation, which provides a label and description in multiple languages as well as the instances and specialisations for the selected concept. As a developer, you can reuse this knowledge – which may help users to understand visualisation terms – instead of providing it on your own. Beyond this simple usage, you could also benefit from the rankings offered by the *VISO/facts* module to suggest appropriate graphic relations for your data.

Thanks to the collaboration with another visualisation project [VPGM12] worked on by Voigt et al., all VISO terms were intensely discussed between two scientific groups right from the beginning. Since their project focuses on different aspects of visualisation, the fields touched by VISO are comprehensive. Therefore, beyond its benefit for the interoperability of visualisation systems, we think that our ontology can serve the visualisation community as a foundation to further formalise, align and unify its knowledge. By clarifying synonyms, homonyms, and overlap of concepts, it supports a common understanding between all interdisciplinary stakeholders in the visualisation process. Further, VISO could be beneficial to classify newly developed graphic representations, publications and other work of the visualisation community. This would ease the search for visualisations and papers as well as the discovery of research directions that received little attention in the past.

5.11 VISO on the Web

An initial version of this ontology is published at the URL *http://purl.org/viso/* to document it and share it for the discussion within the community. A detailed documentation of each ontology module is automatically generated using the LODE ontology-documentation tool[12] and includes a description of each ontology term, its related terms, and depictions. However, the ontology and its documentation are not meant to be in a final state, but to be a basis for ongoing discussion. We can only benefit from the aspect of a *shared* ontology, if it is broadly used and accepted by the community. Therefore, we created a web-platform with rich communication and voting abilities to allow for discussing each ontological term. The platform, which is tightly integrated with the ontology documentation, allows for both ranking existing definitions and proposing new interpretations. Because the ontological terms can only be effectively discussed if the original literature sources are available, we further extended the documentation to show examples as well as quotations and literature references for each resource (Fig. 5.16). Since quotations, references, and authors are modelled in RDF as well, they can be conveniently queried using SPARQL, e. g., to find out which terms a specific author has influenced.



Composite Graphic Object^c

http://purl.org/viso/graphic/Composite_Graphic_Object

A graphic object consisting of one or more other composite or elementary graphic objects arranged in a graphic space.

Quotations from literature:

- »A composite graphic object consists of a graphic space that contains a set of graphic sub-objects. A graphic sub-object may be a composite graphic object itself, or it may be an elementary graphic object.« Jörg von Engelhardt , "The language of graphics", 2002 ,p. 14

has super-classes
- Graphic Object^c

has sub-classes
Composite Symbol^c

Figure 5.16: Excerpt of the VISO documentation showing the term *Composite Graphic Object*. Quotations from literature are given as annotations. Each term can be commented and discussed.

[12] http://lode.sourceforge.net, accessed: 02.07.2015.

Chapter 6

A VISO-Based Abstract Visual Model – AVM

The Abstract Visual Model (AVM) abstracts from concrete platform dependent models such as HTML or X3D. It can be seen as a *platform-independent model* (PIM) in the sense of the Model Driven Architecture[1] (MDA). In contrast to scenegraphs, which focus on offering a model for rendering the geometry of a scene, including logical and physical constraints, the AVM offers a model for information visualisation graphics that explicitly stores the graphic relations of graphic objects. This allows for using the AVM for calculations concerning (additional) valid mappings. Concrete positioning in space is only stated, if information is directly mapped on position in a graphical space. This also means that no layout information is stored and tools interpreting the AVM have to use layout algorithms. The model does not necessarily form a tree structure, but may form an arbitrary graph. The AVM is made up from terms of the VISO. It is an RDF model with instances of type viso-graphic:GraphicObject, related to each other via viso-graphic:GraphicObjectToObjectRelations in order to form visual structures. Each viso-graphic:GraphicObject may also have a set of viso-graphic:GraphicAttributes attached defining properties of the object in the graphic space such as position or colour. Fig. 6.1 shows an example of an abstract visual model of two graphic objects being visually linked to each other, each graphic object representing a resource in the (domain) RDF model.

In the remainder of this chapter, we have a closer look at the »building blocks« of the AVM – graphic objects, graphic attributes and other graphic relations. Furthermore, we introduce a role-based approach to compose graphics on a fine-grained level, describe composition classes based on these roles and give examples of AVM models. Finally, we discuss why there is the necessity for an additional abstraction like the AVM.

6.1 Graphical Notation Used in this Chapter

Since displaying the whole RDF model of an AVM graphically leads to very crowded and complex diagrams, throughout this chapter, we use a specific graphical notation for AVM, which we developed for illustration purposes. This notation does not scale well to arbitrary models, though. Fig. 6.2 shows the model of Fig. 6.1 using this notation. The pentagons represent graphics objects, labeled with their shapes (circle, line). Rhombs stand for relationships; played roles are represented as rounded rectangles. The AVM could be seen as a further application of the *Object Role Model* (ORM) of Halpin [Hal01]. Consequently, the graphical notation of ORM [Hal05] may be a candidate notation to substitute the notation that we use in this

[1] Model Driven Architecture. http://www.omg.org/mda/, accessed: 12.12.2015.

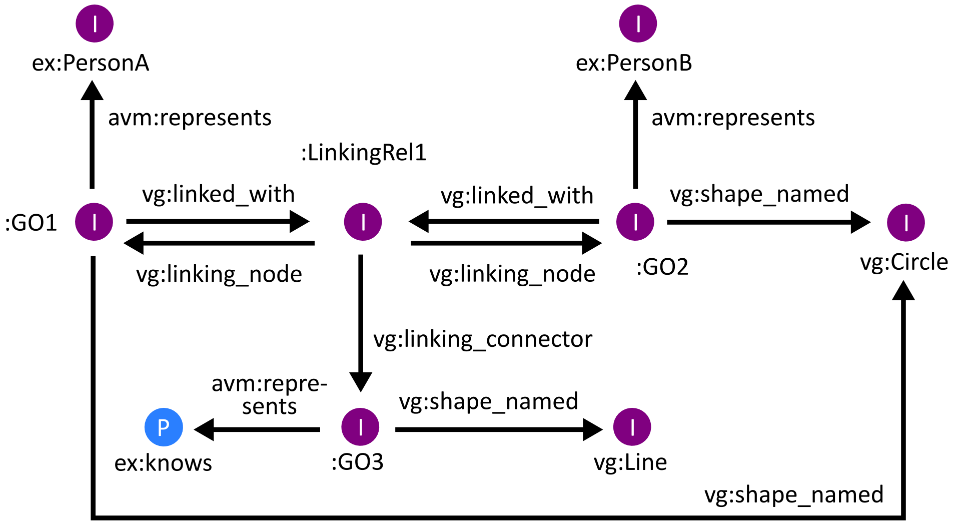


Figure 6.1: Example of an Abstract Visual Model (AVM) describing a complex graphic object with concepts and relations defined in the *VISO/graphic* ontology: The graphic relation viso-graphic:linked_with is used to relate various instances of viso-graphic:Graphic_Object. Notation: I for instance, P for property, cf. legend in .

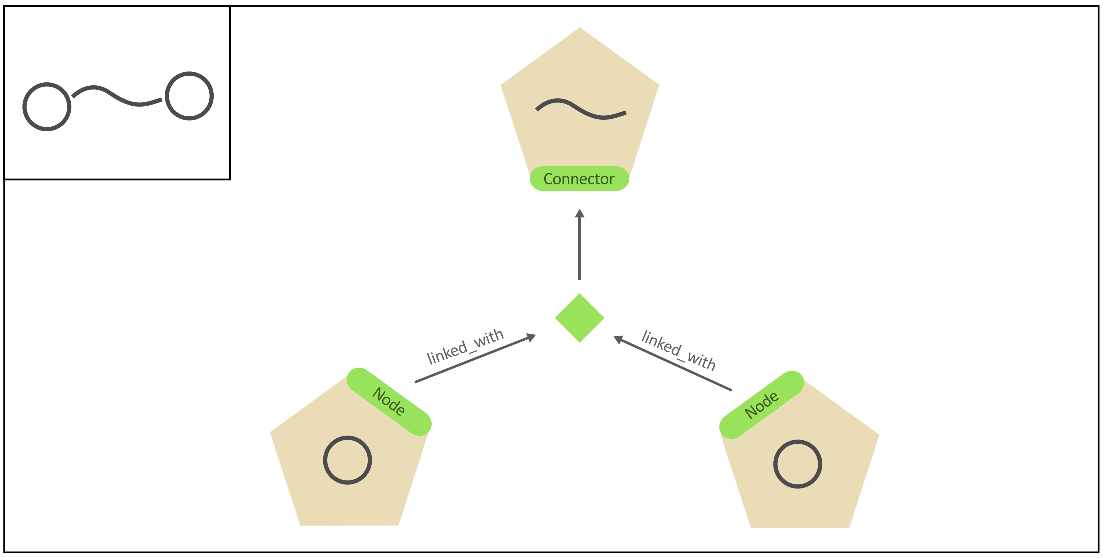


Figure 6.2: The same AVM as shown in Fig. 6.1 using a graphical role notation (hiding some information such as links to the represented resources). In the upper left corner a concrete graphic implementing the model is shown.

chapter for the modelling of graphic roles, which could be rather seen as an ad hoc extension of entity–relationship diagrams (Chen's notation, [Che76]).

6.2 Elementary Graphic Objects and Graphic Attributes

Elementary graphic objects carry the graphic attributes such as viso-graphic:shape_named or viso-graphic:color_hsl_lightness. Furthermore, they can – but do not have to – represent RDF nodes from the domain data, which can be stated in the AVM as well. Elementary graphic objects will be referenced in graphic relations, which can be binary or n-ary.

6.3 N-Ary Relations

Most graphic relations are n-ary, since either an additional graphic object (e. g., a *connector* or a *separator*) is part of the relation, or a quantitative value needs to be added (weighting), such as the amount of overlap for the *superimposition* relation.

For implementing n-ary relations in RDF, the n-ary relation design pattern[2] (Pattern 1) can be used. The pattern suggests creating an individual for each relation and relate two objects via this individual. In our case, the relations from this individual to the participating resources exactly correspond to the roles that the involved graphic objects are playing.

6.4 Binary Relations

Only a few graphic relations are binary, among those simple forms of the well-known »containment« and »links to« relation. For implementing these relations in RDF, it would have been possible to use a single property (e. g., »contains somehow« or »links to somehow«) to point from the containing graphic object to the contained graphic object. However, the roles of the two concerned resources (e. g., »container« and »containee«) cannot be explicitly named, when proceeding like this. Furthermore, even seemingly simple graphic relations – which seem to be binary at first glance – involve extra graphic objects, which turn them into n-ary relations. For this reason, we treat binary relations as n-ary relations.

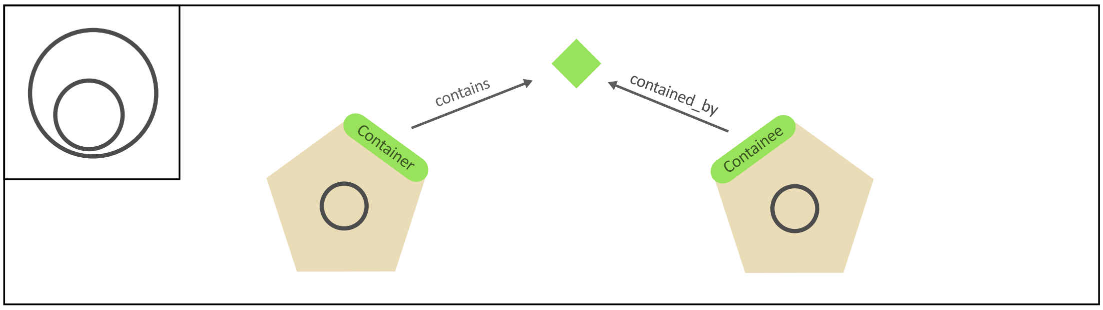


Figure 6.3: AVM of a *containment relation* between two graphic objects. In the upper left corner, a concrete graphic implementing the model is shown.

[2] http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/, accessed: 02.07.2015.

6.5 Composition of Graphic Objects Using Roles

In the above explanation of the AVM, we used the term »role« a few times already. We use »role« in the meaning of a »syntactic role«, which graphic objects can play according to Engelhardt:

> » A syntactic role is a role that a graphic object may play within a syntactic structure. We distinguish [...] node, label, connector, separator, container, point locator, line locator, surface locator, volume locator, metric bar, and grid line.
>
> *Jörg von Engelhardt [vE02]* «

Unlike Engelhardt, who uses roles to *analyse* graphics and describe their *decomposition*, we try to use roles for *synthesising* and *composing* graphics. For example, we do not model graphic objects to be of a certain type, i. e., we do not model: »GraphicObjectA *has type* Connector«; but we state: »GraphicObjectA *plays role* connector«. The use of roles in the AVM was not in first place motivated by the work on modelling with the role concept in software engineering as described by Steimann [Ste00] but was directly motivated by the work of Engelhardt. Nevertheless, of course, the way we use roles in the AVM could be classified using the work of Steimann as »roles as named places [of a relationship]«.

Being able to reference graphic objects via their role enables us to compose complex graphic objects from elementary ones on a very fine-grained level, as opposed to the high-level composition of complete graphic representations (e. g., in a mash-up scenario). Examining the roles of graphic objects also helps us to classify these compositions. We define two criteria for classifying compositions of graphic objects:

Single vs. Multiple Roles A graphic object can play a *single* or *multiple roles* at a time

Same vs. Different Role Type A graphic object can play multiple roles of the *same type* or *different types*

Fig. 6.5 shows the three composition cases emerging from this: The first row represents a very simple artificial composition case: Each graphic object plays only a single role at any point in time. Graphic objects have to be duplicated to take part in more than one graphic relation. The second row represents a slightly more complex composition case – it is now valid for a graphic object to play a graphic role more than once, but it has always to be the same role. In the *directed linking* example (second column), *A* plays the role of an »end node« twice. In the third row, we allow for graphic objects playing multiple *different* roles at the same time. Graphic object *B* plays both the role of a »container« and a »containee«, respectively for linking, *B* plays the role of »start node« and »end node« at the same time.

6.6 Composition of Graphic Relations Using Roles

A further criterion for classifying graphics is whether multiple *different* graphic relations are involved in the same graphic, which was not the case in the examples above. Fig. 6.6 demonstrates this at the example of *(undirected) linking* and *containment*. Two of the nodes (»node« is a role assigned by the graphic relation »linked_to«) simultaneously play a role in a »containment« relation. From here on, in addition to a composition of graphic objects, we have a composition of graphic relations:

Same vs. Different Graphic Relation A graphic object can play multiple roles belonging to one and the *same graphic relation* or belonging to *different graphic relations*, i. e., it can take part in different graphic relation types.

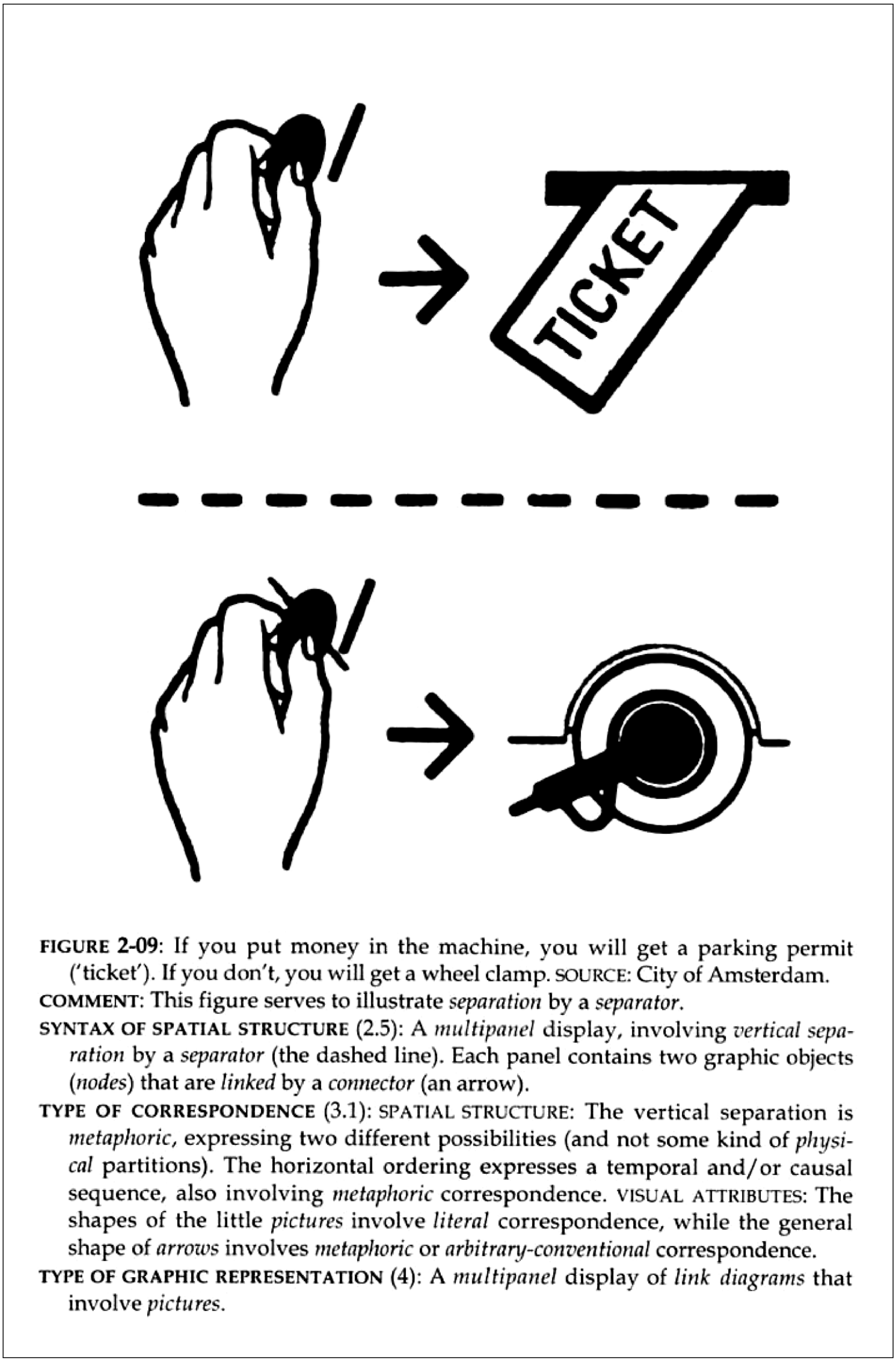


FIGURE 2-09: If you put money in the machine, you will get a parking permit ('ticket'). If you don't, you will get a wheel clamp. SOURCE: City of Amsterdam.

COMMENT: This figure serves to illustrate *separation* by a *separator*.

SYNTAX OF SPATIAL STRUCTURE (2.5): A *multipanel* display, involving *vertical separation* by a *separator* (the dashed line). Each panel contains two graphic objects (*nodes*) that are *linked* by a *connector* (an arrow).

TYPE OF CORRESPONDENCE (3.1): SPATIAL STRUCTURE: The vertical separation is *metaphoric*, expressing two different possibilities (and not some kind of *physical* partitions). The horizontal ordering expresses a temporal and/or causal sequence, also involving *metaphoric* correspondence. VISUAL ATTRIBUTES: The shapes of the little *pictures* involve *literal* correspondence, while the general shape of *arrows* involves *metaphoric* or *arbitrary-conventional* correspondence.

TYPE OF GRAPHIC REPRESENTATION (4): A *multipanel* display of *link diagrams* that involve *pictures*.

Figure 6.4: Syntactic roles in Engelhardt's work [vE02]: Engelhardt analysed many graphics by decomposing them and describing the syntactic roles that the graphic objects play – in this graphic *separator* and *connector*. Image taken from [vE02].

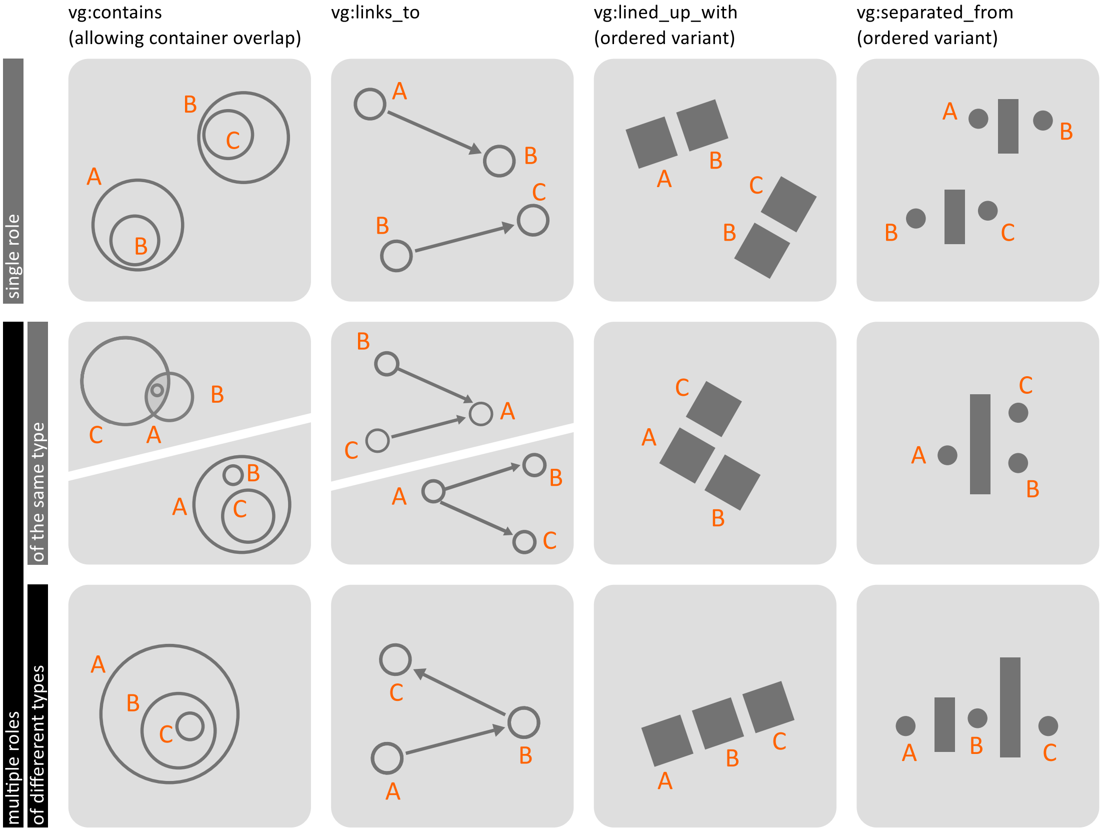


Figure 6.5: Classification of the composition of graphic objects by roles.

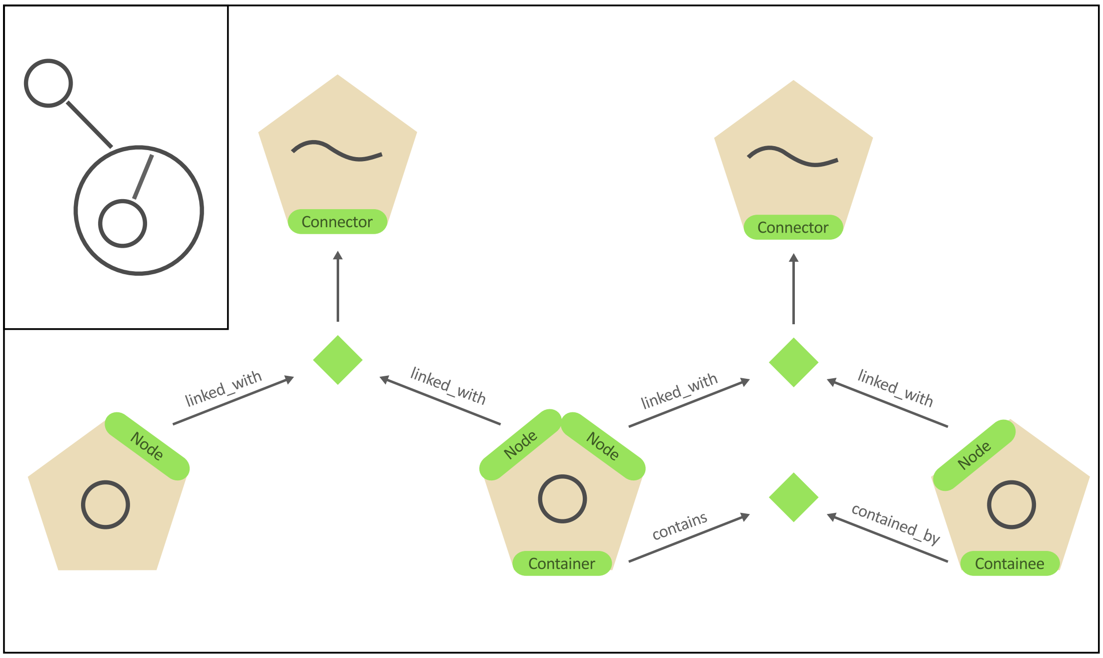


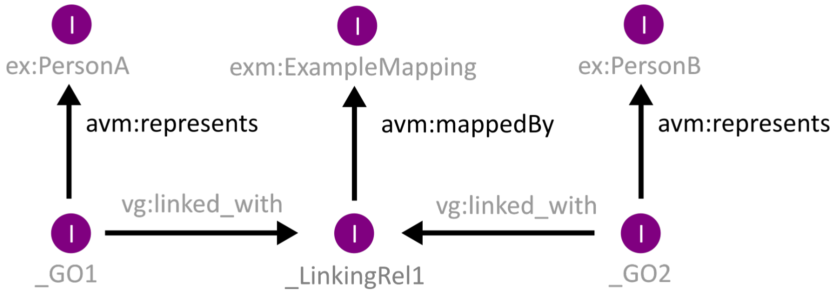
Figure 6.6: Example of an AVM where two graphic relations are used in combination: The centre and the left »node« simultaneously plays the role of a »container« respectively a »containee«. In the upper left corner a concrete graphic implementing the model is shown.

6.7 Composition of Visual Mappings Using the AVM

The AVM is also the foundation for our approach to the composition of visual mappings, which will be introduced in Sect. 7. Similar to the composition of graphic objects and relations, roles in the AVM serve as an anchor for the composition of visual mappings. Hence, we revisit and extend the classification of compositions in Sect. 8.2.1. Furthermore, the AVM allows for calculating composable mappings. Some visual mappings can only jointly be applied if certain conditions are met, e. g., there might be constraints on already defined mappings and on the data that shall be mapped (cf. Sect. 5.7.3). This would be impossible, if we only stored a final graphic, even in formats like SVG, since for the calculation of further applicable mappings, we need to know a) which mappings were already applied and b) which parts of the graphic resulted from which mapping. Checking for constraint violations and the resulting guidance activities that become possible on the level of the AVM are explained in Sect. 8.1. Thus the AVM – as a common target format – is the prerequisite for composing visual mappings in the OGVIC approach. It offers the opportunity to store visual mappings in a reusable form.

6.8 Tracing

Calculations on the composability of mappings, as described in the last section, require tracing capabilities. The introductory AVM example already showed references from graphic objects to the resources they represent. The following excerpt of an AVM model uses two kinds of such trace links – avm:represents and avm:mappedBy – to point to the data and the visual mappings:



The vocabulary used for this purpose belongs to an extra module of VISO – the *VISO/avm* vocabulary[3]. Using the property avm:represents, we can trace back to the origin of a graphic object, i. e., the rdfs:Resource or rdfs:Literal that was mapped to the graphic object. Similarly, avm:mappedBy points from (n-ary) graphic relations to the mapping that created them.

6.9 Is it Worth Having an Abstract Visual Model?

The additional model introduced with the AVM can be justified for three reasons. First, platform-variability *(R-10)* requires such a model. If we want to define visual mappings independently from the final platform, an abstraction has to be created, which then becomes refined to multiple concrete graphic formats containing the platform-specific settings. Platform-variability is worthwhile in the following two cases:

a) There are graphic relations that belong to a common core of graphic relations available on each platform (except for naming differences)

[3] http://purl.org/viso/avm

b) There are graphic relations that can be defined abstractly and refined to a concrete setting for each platform

However, it has to be considered that a concrete platform may not support all graphic relations and, hence, not all types of graphics that may have been defined. There are dependencies between concrete output platforms and the available types of graphics. For example, HTML cannot display Node-Link-Diagrams. That means concrete platforms and visual structures (or »Representation paradigms«, the term used in Fresnel) are two aspects that are not fully orthogonal. An initial idea to solve this issue was to define only the intersection of those graphic relations in the AVM that are available on all concrete platforms. Since the set of potential output platforms is unknown, this is not possible, though. Instead, for such cases, alternative solutions should be definable. As an example, consider a platform that does not support colour. On this platform, an alternative mapping could be defined to replace colour values by grey scale values. Further examples are:

- Colour $\mapsto$ Texture
- 3D $\mapsto$ 2D Small Multiples
- Animated 2D $\mapsto$ 2D Time-Series

Similarly, graphic relations that cannot be represented due to missing graphical power could be replaced by falling back to alternative ones. There are two possibilities to define alternative mappings for replacing graphic relations not available on a given platform:

a) Define alternative mappings for the platform in general

b) Define alternative mappings only for a specific data set

The RVL mappings that will be introduced in the next chapter, support (b) with the *fallback* mechanism (Sect. 7.2.1). Another observation is that the overlap between concrete platforms is reduced by the efforts to enable separation of concerns. For example, HTML and SVG both use CSS. Conceptually, all overlap between HTML and SVG (concerning graphic attributes) is treated by CSS, and a single mapping to CSS is now sufficient in theory. In practice, different names may be used in CSS for SVG and HTML, e. g., »background-colour« vs. »fill«. Only the structural information may still overlap, since both HTML and SVG can create nested boxes, list and tables, for example. This means, with the use of CSS, platform-independence mostly concerns the structural aspects of graphics.

The second reason that justifies the additional abstraction introduced by the AVM are composition *(R-8)* and guidance requirements *(R-15)*. As discussed in Sect. 6.7, we need to analyse not only the existing mappings, but also the preliminary graphic, so that guidance tools can recommend valid additional mappings and answer questions like »Does it make sense to add a mapping X from the visual perception point of view?«. The AVM, which explicitly models both graphic relations and references to mappings, enables powerful introspection options for this purpose. This would not be possible, if we directly generated a graphic by means of the final platform such as SVG. Also interaction and editing require diagrams to be “internally represented by a formal model” [Min00] (see also Sect. 6.11). In Table 6.1 we compare the AVM from Fig. 6.1, which represents a line connecting two nodes, to a possible SVG »implementation« of this AVM. Metadata describing the document can be embedded as RDF in SVG as recommended in the SVG specification[4]. So the information on which graphic object represents which resources could possibly be embedded directly into SVG as well. However, it also becomes apparent that there is no explicit information on how the three graphic objects are visually related in the SVG. Each object is independently positioned in the graphic space by means of absolute coordinates.

[4] SVG 1.1 (Second Edition) — 16 August 2011. W3C Recommendation – http://www.w3.org/TR/SVG11/metadata.html

A third reason for the AVM is reusability *(R-7)*. Currently, for the definition of visual mappings, each application uses its own language, although approaches such as ViZmL and GPL have been proposed. For RDF-based data, there is no such language at all. Therefore, reusability of visual mappings across platforms and for various visual structures is a further argument to justify the AVM.

6.10 Discussion of Fresnel as a Related Language

The RDF display vocabulary Fresnel served as inspiration for the RVL language, which we discuss in detail in Chapter 7. The language was intentionally designed to be independent of platform and visual paradigm for reasons of reusability of display knowledge (cf. Sect. 4.3.2). Fresnel does not describe a rendering to a concrete platform such as HTML, SVG or X3D, but introduces an additional level of abstraction – similar to the introduction of the AVM in this thesis. But although Fresnel was presented already in 2006, it is still not widely used and did not become a standard. Therefore, in this section we look at experiences with Fresnel and discuss critical statements on Fresnel and the level of abstraction that Fresnel adds to the visualisation process. We conclude that our approach of creating an AVM is not harmed by criticism on Fresnel. Fresnel has been criticised for the following reasons[5]:

Too complicated – The benefits of Fresnel are not worth the additional effort of learning another language.

In [RCC09], the authors of VPOET find that the syntax of Fresnel requires »*skills in semantic web technologies that severely limit the number of designers available*«. However, it is not a requirement that Fresnel is written manually. With an appropriate (graphical) UI, Fresnel descriptions could be created conveniently by domain experts as well. Also the developers of Tal4RDF see it as beneficial to directly aim at rendering the presentation format. However, they also see the option that their approach could possibly complement Fresnel: »*an implementation of Fresnel could rely on T4R for the actual rendering.*«[6]

Too abstract – Too much formatting decisions remain to the rendering engine.

From the examination of platform and visual structure independent features of Fresnel in Sect. 4.3.2, we conclude that Fresnel defines a presentation vocabulary that leaves many decisions to the renderer, but still offers a core of clearly platform-independent presentation features. This core seems to justify platform-independence. Nevertheless, for example the authors of OWL-PL argue that »[. . .] *the formatting options available to the author are too generic as a result, and individual implementation is left up to the application incorporating Fresnel. This may remove too much control from the author's hands when deciding how to display their data*« [BH10]. OWL-PL simply overcomes these problems by forgoing the criterion of platform-independence. While this may be disputable for creating documents, this is not an option for storing arbitrary visual mappings. The authors of Fresnel are aware of the limited expressivity, but state that the expression of more knowledge would cause the vocabulary to loose its characteristic of being paradigm-/platform-independent.

Replaceable by existing technology – Fresnel adds little value beyond XSLT, CSS, and SPARQL.

[5] Much of the criticism concerns the abstraction introduced by Fresnel, rather than concrete language constructs. Therefore, and to have all the discussion in one place, we decided to discuss Fresnel in this chapter, not in the next one on the RVL language.

[6] Tal4RDF documentation. http://liris.cnrs.fr/~pchampin/t4r/doc/rationale.html, accessed: 05.02.2011.

```
@prefix : <http://purl.org/rvl/example-avm/> .
@prefix avm: <http://purl.org/viso/avm/> .
@prefix vg: <http://purl.org/viso/graphic/> .
@prefix common-shapes:
       <http://purl.org/viso/shapes/common/> .
@prefix ex: <http://example.org/> .

:LinkingRel1
       a vg:Linking_Undirected_Relation ;
       vg:linking_connector :GO3 ;
       vg:linking_node :GO1 , :GO2 .

:GO1
       a vg:Graphic_Object ;
       vg:linked_with :LinkingRel1 ;
       avm:represents ex:PersonA ;
       vg:shape_named common-shapes:Circle .

:GO2
       a vg:Graphic_Object ;
       vg:linked_with :LinkingRel1 ;
       avm:represents ex:PersonB ;
       vg:shape_named common-shapes:Circle .

:GO3
       a vg:Graphic_Object ;
       avm:represents ex:knows ;
       vg:shape_named common-shapes:Line .
```

```
<svg

  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:cc="http://creativecommons.org/ns#"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns="http://www.w3.org/2000/svg"
  [...]
  version="1.1" >

 <metadata id="metadata237">
   <rdf:RDF>
     <cc:Work>
       <dc:format>image/svg+xml</dc:format>
       <dc:type rdf:resource="http://purl.org/
                dc/dcmitype/StillImage" />
     </cc:Work>
   </rdf:RDF>
 </metadata>

 <path id="path871"
   d="m 17,25 85,-9"
   style="fill:none;stroke:black" />
 <circle id="ellipse240"
   r="10" cy="16" cx="102" />
 <circle id="ellipse241"
   r="10" cx="17" cy="25" />

</svg>
```

Table 6.1: Comparison of AVM and SVG. Both listings describe the graphic from Fig. 6.1, which represents a line connecting two nodes. The left column shows the AVM model in Turtle syntax. The right column shows a possible SVG representation of the described graphic (incl. exemplary metadata on the SVG document).

Another critique of Fresnel is that it could be almost completely replaced by reusing »*technologies like (X)HTML, XSLT, and CSS*« [BH10]. One might also argue that the features listed in Sect. 4.3.2 are purely data-related and languages such as SPARQL should be used instead. But, although SPARQL allows for the selection of resources and the ordering of values, SPARQL neither offers a means to store »views« created by a SPARQL-query, nor allows for attaching them to resources. Additionally, some selections may more easily and concisely be stated by a path language such as FSL (Fresnel Selector Language, Sect. 4.3.1) than with SPARQL[7]. For reasons discussed in Sect. 4.3.1, languages such as XPath (used by XSLT) cannot simply be applied to RDF.

A last issue that we would like to raise is the question of **flexibility**, because this also applies to our approach of generating an AVM. It is hard to define **additional formatting** in case the formatting defined with Fresnel is not sufficient. A reduction of the visualisation designer's flexibility is the drawback of all approaches that use abstract platform-independent models to generate platform-specific ones. Although additional formatting could be defined manually, the changes will be lost when regenerating the platform-specific code.

6.11 Related Work

While the AVM is novel implementing Engelhardt's syntactic graphic roles and being located in the RDF technical space, there are approaches from the graph transformation community that use graphs as formal, internal models for diagrams in order to enable tool support for visual languages (for example to build diagram editors). Therefore, we need to point to commonalities and differences of their approaches:

Bardohl et al. [BTMS99] discuss various approaches, beginning with the early approach of Rekers and Schürr [RS96] (as cited in [BTMS99]), who use a *Spatial Relation Graph* (edges represent graphic relations, nodes represent graphic objects). This resembles the »binary version« (Sect. 6.4) of modelling graphic relations, which we initially also considered for the AVM but abandoned, since most graphic relations could not be handled. Minas [Min00] uses hypergraphs to provide a uniform, internal model of a diagram and shows that hypergraphs cannot only model node-link diagrams, but arbitrary spatial relations, such as *inside* and *overlaps* (*containment* with and without overlap in our terminology). While we use multiple associated (RDF) properties to allow for modelling complex n-ary graphic relations and roles of graphic objects in the AVM, Minas uses hyperedges and their tentacles to model spatial graphic relations and their *attachment areas*.

Hypergraphs and graph transformations are not widely referenced in the information visualisation literature, but rather familiar to researchers in the field of visual languages and diagram editors, possibly because the data structures to be visualised in these fields are graphs themselves. Since we are visualising knowledge *graphs* – not tabular data, as it is frequently the case in information visualisation – such techniques are interesting for us as well.

6.12 Limitations

The AVM, as described in this chapter, does not allow for tracing from graphic *attributes* back to the resources and data properties that they represent. This also applies to tracing graphic attributes back to the mappings.

[7] SPARQL 1.1 introduced property-paths, though (cf. http://www.w3.org/TR/sparql11-query/#propertypaths).

6.13 Conclusions

In this chapter we introduced the AVM as our approach of a platform-independent model of graphics. We showed what are the constituent parts of this model and why the additional abstraction introduced by such a model is necessary; in summary, because it supports reuse by platform-variability and benefits guidance and composability by enabling tracing and introspection. The necessity for a rather complex modelling approach like the AVM is additionally supported by Minas [Min00], who suggests hypergraphs for handling also the less trivial graphic relations. Future work on the AVM, should carefully compare the work of Minas, also with respect to syntax-checking the AVM with graph grammars.

Furthermore, we compared the abstraction of the AVM to the abstraction that Fresnel introduces, which had been criticised for various reasons although it is frequently cited and was adopted by many projects. Nevertheless, in summary, most of the criticism on Fresnel can either not stand the discussion or does not equally apply to a visualisation language. Concerning the criticised reduction of flexibility, we consider this tolerable for many visualisation scenarios. Where it is not, one option to improve flexibility could be the adoption of ideas from round-trip engineering [Aß03a] to the visualisation process, cf. future work on editing described in Sect. 10.6.

The RVL language presented in the next chapter relies on the graphic roles provided by AVM models to realise mapping composition.

Chapter 7

A Language for RDFS/OWL Visualisation – RVL

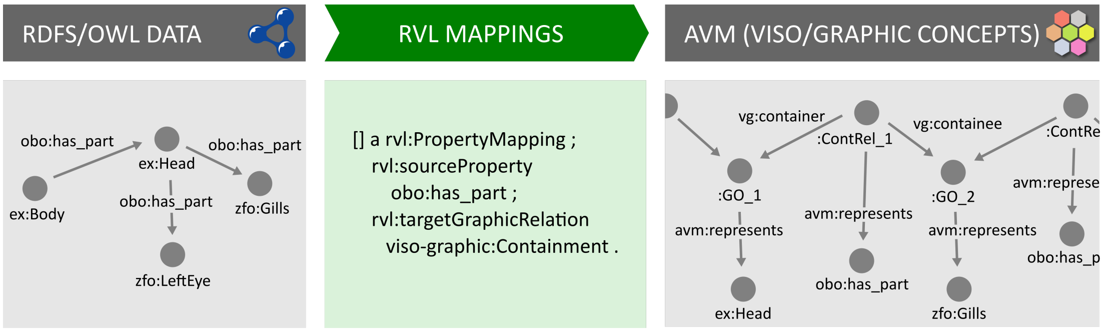
This chapter is based on work published in [Pol13].

Visualising data requires a mapping from data objects to graphic objects. To date, each visualisation tool uses its own internal solution for storing visual mappings, making it impossible to reuse effective settings in different contexts and share them with others. Sharing visualisation settings between users would be beneficial for two reasons: First, tools have different strengths with respect to the handling of large data, interactive analysis features or accessibility. Second, authors of domain ontologies could suggest useful visualisations and publish them along with their data. While there are presentation or display languages for RDF data such as Fresnel[1], these serve the purpose of turning data into formatted documents, but they do not allow for explicitly defining visual mappings. Up to now, no language specific to RDFS/OWL visualisation was published. Hence, during the problem analysis (Sect. 3), we formulated the research question *Q-1*, asking how to define composable, shareable visual mappings from ontological data to visual means and what should be the ontology constructs that can be mapped.

To answer this question, we propose the *RDFS/OWL Visualisation Language* (RVL) as a declarative visualisation language for ontological data. RVL defines a set of mapping types that can be instantiated to describe visual mappings from relations, classes and individual values in the source data to graphic concepts. On the source data side, RVL mappings reference RDF(S)/OWL properties, classes and individuals from the domain data. On the target side, the visualisation mappings point to concepts of the *VISO/graphic* ontology, an ontology on graphic concepts and relations that we introduced in Chapter 5. Fig. 7.1 gives an overview of this general idea at the example of a Property Mapping. By various mapping types and preferences such as Value Mappings, RVL does not only allow for simple one-to-one-mappings of single source values to single graphic values, but offers rich capabilities to steer the calculation of mappings. Assuming defaults, RVL still allows for quickly handling common mapping situations.

In the following, we briefly sum up problems of current approaches to visual mappings for RDF data and recall the list of common visualisation cases that we identified during our case studies. Based on this, we establish a set of concrete language requirements that a novel language for RDFS/OWL visualisation must fulfil (Sect. 7.1) and discuss choices in language design as

[1] Compare for Sect. 4.3.2 for a detailed comparison of RDF presentation languages, which we did during our analysis of related work.



```
[] a rvl:PropertyMapping ;
   rvl:sourceProperty
      obo:has_part ;
   rvl:targetGraphicRelation
      viso-graphic:Containment .
```

Figure 7.1: The principle of RVL. Mappings in RVL describe visual mappings from relations in the domain data to graphic relations defined by *VISO/graphic*. This is illustrated using a simplified example of a rvl:PropertyMapping, which maps the ontological relation obo:has_part to the graphic relation viso-graphic:Containment_Relation. The result is an Abstract Visual Model (AVM) of viso-graphic:GraphicObjects related by viso-graphic:Containment_Relations (only a fraction of the AVM is shown here). Rendering to a concrete graphic is not in the scope of RVL.

well as the limitations of our language. Afterwards, we present our approach – the *RDFS/OWL Visualisation Language* (RVL) – in Sect. 7.2. We cover the main language constructs, followed by an additional section on the composition of mappings (Sect. 7.10). Sect. 7.11 discusses our choice for defining a schema for RVL and describes how editors for RVL can easily be derived from the rich schema definition. Finally, in Sect. 7.12, we briefly evaluate the current status of RVL with respect to the use cases we defined and point to open issues for future work.

7.1 Language Requirements

During the problem analysis (Chapter 3), we formulated the following problems of current approaches to ontology visualisation: We criticised that ontology visualisations are created for specific platforms and separately for each visual paradigm. Further, we noted that they cannot be shared with others and reused on other visualisation platforms *(P-3)* or in combination with different visual paradigms *(P-4)*. We concluded that domain authors lack a standard to store visualisation settings based on standards *(P-5)*, since a visualisation language for ontological data does not exist *(P-6)*. The problems *P-3 – P-6* are tackled in this chapter. Further, we described the need to cover a set of visualisation cases *(VC-1 – VC-12)*. These cases are referenced in the following discussion of our language requirements.

Based on our general goals and requirements, the concrete visualisation cases we identified, and experiences described in related work, we formulate 14 language requirements (LR) for a visualisation language for RDFS/OWL data:

LR-1 The language must support the visual mapping, not only the presentation of data.

LR-2 The language must support multiple visual structures.

LR-3 The language must allow for the definition of simple interactions.

First, we require RVL – as a *visualisation* language – to describe dynamic and value-dependent visual mappings *(LR-1)* as opposed to just style a document with explicit static settings such as font type or background colour, like it is possible with CSS (Sect. 2.1.6). Instead of focusing on node-link diagrams (graphs), we want to allow for choosing from multiple visual structures such as containment (enclosure) structures *(LR-2)*. Furthermore, also simple interactions with the visualisation should be usable in the mappings such as »What happens on select?« (*LR-3*; *VC-2 to VC-4 and VC-10*).

In many cases, as for *LR1* – *LR3*, the language requirements directly correspond to general requirements that we defined for our approach as a whole (Chapter 3). In other cases, they refine the general requirements to more concrete requirements on the language. The language requirements will also resemble the criteria we set up for comparing presentation languages in Chapter 4.3.2.

LR-4 Ontology terms must be referenceable via their URI.

LR-5 T-Box ontological relations must be referenceable.

Referencing ontology terms such as classes and properties, from both the domain ontologies and the *VISO/graphic* ontology module, needs to be easy in the language. Since all domain ontologies from our case studies as well as *VISO/graphic* are available in RDF, and each of these terms can be identified via its URI as a unique identifier, referring to the ontology terms is quite simple. It only requires that URIs are allowed in the mapping definitions *(LR-4)*. Using common standards such as URIs and W3C-recommended ontology languages based on RDF greatly improves the sharing of the mappings. Further, we require that a specific visualisation of T-Box statements is possible and these relations can explicitly be referenced in mappings *(LR-5)*.

LR-6 Visual mappings must be defined independently of a specific platform.

LR-7 Visual mappings must be defined independently of any specific visual structure.

LR-8 Visual mappings must be defined in declarative style.

LR-9 The language must support sharing visual mappings.

LR-10 Each visual mapping must be identified via a URI.

Since we want to be able to reuse mappings in different visualisation tools, we require the mappings to be platform-independent *(LR-6)*, i. e., the mappings need to be formulated without referring to platfom-specific terms. Similarly, we want to be able to vary the visual structures and we require the mappings to be formulated in a way that is independent from a specific visual structure *(LR-7)*. That means, mappings should be independently replaceable as far as possible. Due to the requirements *LR-6* and *LR-7*, we further require RDF to be declarative *(LR-8)*, i. e., the language should describe *what* should happen as opposed to *how* exactly it should happen. Arguments for a declarative style can be found in the discussion on both style sheet languages such as CSS [Lie05] (cf. Sect. 2.1.7) and visualisation languages such as Protovis [HB10] (cf. Sect. 4.2.1). For Protovis, Heer and Bostock argue that by using a declarative language »[. . .] *visualizations become open source: since specifications are concise and are not compiled, but instead interpreted at runtime by the web browser, users can easily view the source code and data behind any visualisation. In addition to learning by example, visualisations are constructed from modular primitives, which make it easier for designers to incorporate discovered techniques into their own work through copy-and-paste. Open source also facilitates some degree of collaboration, since users can more easily create derivative works to show different views or fix mistakes.*« [BH09]. Having decided on a declarative approach still leaves the choice between a transformational approach (e. g., XSL) and an approach where the mappings are interpreted by the client (e. g., CSS). The disadvantages of the transformational approach, such as a loss of semantics, have been listed in Sect. 2.1.7, where we summarised lessons learned from style sheet languages. Consequently, for the design of RVL, the interpretative, CSS-like approach is preferred to a transformational approach. Both, the ability to trace back to the semantics of the source data as well as conciseness and open-source-friendliness are important criteria for RVL as well, and they directly contribute to the general goal of explicit and reusable mappings. Fostering reusability, we need to support shareability *(LR-9)* with our language. The prerequisite for sharing mappings and sending them to other users is that mapping definitions can be named and a concrete syntax exists to describe them. Going one step further, using URIs for each mapping *(LR-10)* even allows us to universally identify a mapping and let it be globally referenced and reused.

LR-11 The extension of visual mappings must be supported.

LR-12 The composition of visual mappings must be supported.

LR-13 The language should be familiar to users.

LR-14 Useful defaults should be specified for the language settings wherever possible.

Only if users can extend existing mappings, according to their specific needs, or combine them with other existing mappings, they can fully benefit from existing work without being constrained in their flexibility (*VC-7 and VC-8*). Therefore, also the extensibility *(LR-11)* and composability *(LR-12)* of defined mappings contributes to reusability. To increase the familiarity to users *(LR-13)*, RVL should resemble other languages from the Semantic Web or InfoVis communities. Ideally, it should also resemble other configuration settings and languages used in our approach (OGVIC). Finally, RVL should specify reasonable defaults *(LR-14)* for common mapping situations such as labelling objects *(VC-12)*.

Limitations

There will be no support for editing mapped data in RVL. While we design the visualisation language to be well usable by tooling in order to allow for convenient editing of the visual mapping definitions, we do not aim at editing support for the source data. Further, RVL will not allow for defining styles directly. Although we aim at a language for visual mapping, custom styling of graphical elements in a graphic representation may be required to describe a visualisation, e. g., for reasons of corporate design, personal preferences or better readability *(VC-11)*. For defining these styles, CSS could be reused, since it is a widely accepted standard. While the full integration of RVL and CSS is beyond the scope of this thesis, RVL should be designed to easily integrate with declarative style sheet languages like CSS, which is another argument for a declarative approach.

7.2 Main RVL Constructs

Having described the general idea of RVL mappings, we now look at the various mapping types we created, relate them to the visualisation cases (VC) we identified (Sect. 3.6) and discuss why we decided to use extra mapping types in some cases while, in general, aiming at as few mapping types as possible. The class diagram Fig. 7.2 gives an overview of important RVL classes. In the following sections, we can only point to the most important principles of RVL. For a complete specification of the language we point to the language reference[2]. For each class shown in the following diagrams of RVL classes (except Fig.7.2), important properties are listed like attributes in the UML. The coloured rectangle in front of each attribute marks whether the property is an owl:ObjectProperty (blue), owl:DatatypeProperty (green) or other (purple). The listings in this chapter are given in Turtle[3] notation.

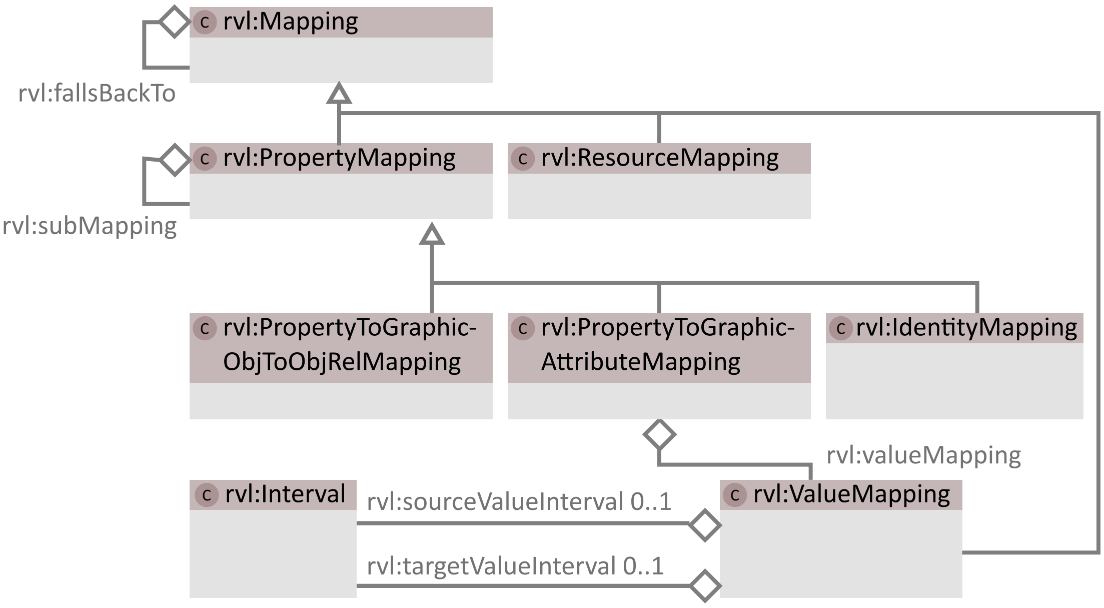
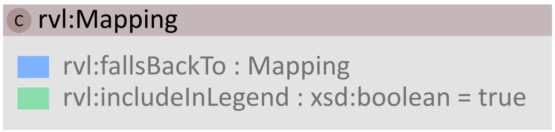


Figure 7.2: Overview of main RVL classes including rvl:PropertyMapping and rvl:ValueMapping following the notation of UML class diagrams.

7.2.1 Mapping

All mapping types in RVL inherit from rvl:Mapping in order to define the appearance of the mapping in a legend (rvl:includeInLegend), which RVL tools should generate by default *(VC-12)*. Additionally, for each mapping, a »fallback« can be defined (rvl:fallsBackTo) to specify what should happen if an RVL tool cannot handle a mapping. For example, a mapping to *colour* could fall back to a mapping to *texture* on black-and-white systems.


c rvl:Mapping
rvl:fallsBackTo : Mapping
rvl:includeInLegend : xsd:boolean = true

[2] http://purl.org/rvl/

[3] Terse RDF Triple Language. http://www.w3.org/TR/2012/WD-turtle-20120710/, accessed: 12.12.2015.

7.2.2 Property Mapping

rvl:PropertyMapping defines a mapping from one domain property (i. e., a domain relation) to one graphic relation. Property mappings apply, whenever a statement using the property (i. e., a property instance) exists, no matter what is the object or subject of the statement. A simple Property Mapping that maps the value cito:cites to Linking_Directed from the viso-graphic vocabulary may be as simple as follows:

```
[] a rvl:PropertyMapping ;
  rvl:sourceProperty cito:cites ;
  rvl:targetGraphicRelation viso-graphic:Linking_Directed_Relation .
```

If constraints need to be made on the application of the mapping, filters have to be applied on the subject and/or the object. These filters can be defined by expressions in FSL (Sect. 4.3.1) or in SPARQL. They will reduce the set of statements to be mapped to those where the object, respectively the subject, matches the filter. Fig. C.1 in the appendix gives an overview of how filters can be used in RVL and what is the difference between subject and object filters.

Two subtypes of Property Mapping exist, one for mapping to graphic attributes *(VC-2)* and one for mapping to Graphic-Object-to-Object-Relations *(VC-3)*. Applying a Property Mapping implicitly creates graphic objects for the involved (domain) resources *(VC-1)*.

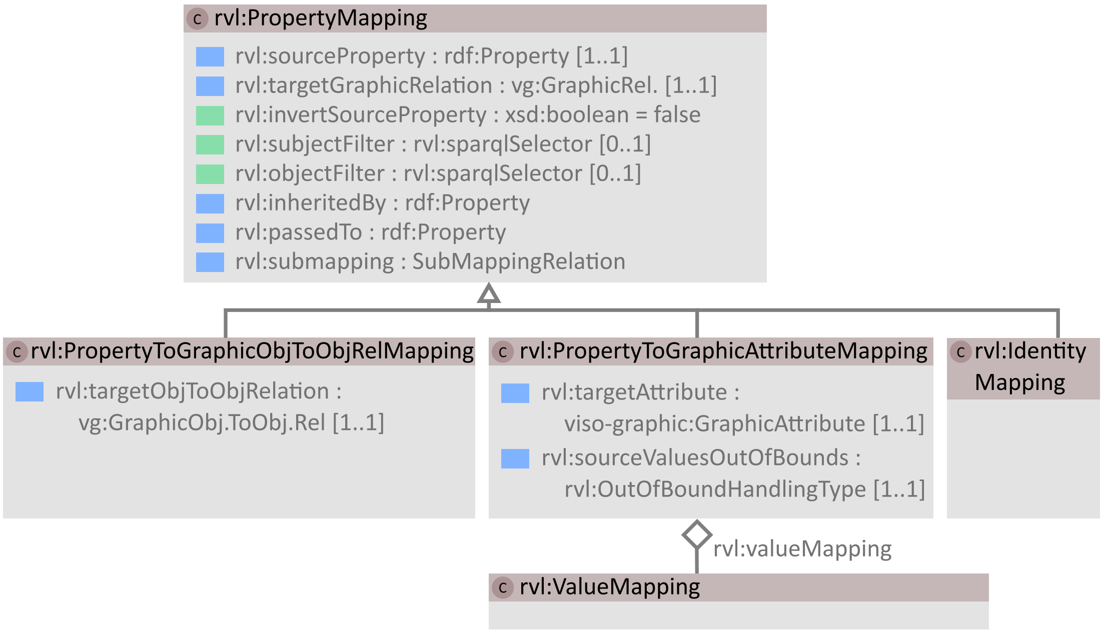

c rvl:PropertyMapping
rvl:sourceProperty : rdf:Property [1..1]
rvl:targetGraphicRelation : vg:GraphicRel. [1..1]
rvl:invertSourceProperty : xsd:boolean = false
rvl:subjectFilter : rvl:sparqlSelector [0..1]
rvl:objectFilter : rvl:sparqlSelector [0..1]
rvl:inheritedBy : rdf:Property
rvl:passedTo : rdf:Property
rvl:submapping : SubMappingRelation
c rvl:PropertyToGraphicObjToObjRelMapping
rvl:targetObjToObjRelation : vg:GraphicObj.ToObj.Rel [1..1]
c rvl:PropertyToGraphicAttributeMapping
rvl:targetAttribute : viso-graphic:GraphicAttribute [1..1]
rvl:sourceValuesOutOfBounds : rvl:OutOfBoundHandlingType [1..1]
c rvl:Identity Mapping
rvl:valueMapping
c rvl:ValueMapping

Figure 7.3: Overview of the rvl:PropertyMapping class.

7.2.3 Identity Mapping

rvl:IdentityMappings pass on the value of the source data and directly use it as the graphic value. A common case for this is to pass on rdfs:label as a text value (shown in the listing below). Sometimes, graphic values may already exist in the source data such as RGB colour values, which can directly be used in the graphic. Identity Mappings are also used in RDFScape [Spl08] and referred to as *Passthrough mappings*.

```
[] a rvl:IdentityMapping ;
  rvl:sourceProperty rdfs:label ;
  rvl:targetGraphicRelation viso-graphic:text_value .
```

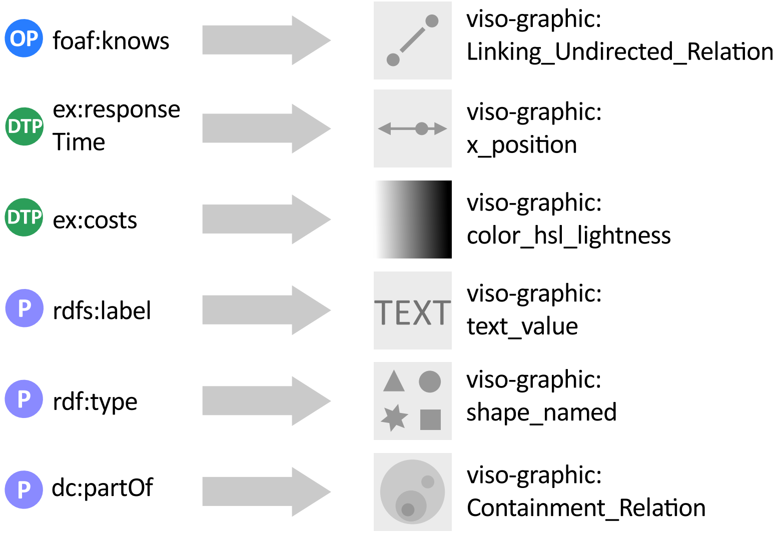


Figure 7.4: Examples of Property Mappings. Although most frequently owl:ObjectPropertys (blue) will be mapped to Graphic-Object-to-Object Relations and owl:DatatypePropertys (green) will be mapped to Graphic Attributes, other constellations are possible. Furthermore, often relations are only modelled as rdf:Property (purple). The fourth mapping (rdfs:label to viso-graphic:text_value) is an example of an rvl:IdentityMapping.

7.2.4 Value Mapping

rvl:ValueMappings define which source values are mapped to which target values. Optionally, they provide further information on how to do this mapping of values exactly. That way, they are a supplement to Property Mappings, which only draw the general connection between a source relation and a target Graphic Attribute[4]. If no Value Mapping is defined, the default value mapping is used (Sect. 7.7). Value Mappings can be seen as shorthand, eliminating the need to define many Property Mappings with simple filters on the object (value) side of a statement. A simple Value Mapping that maps the value ro:Requirement to the shape Square from the common-shapes collection may be defined as follows:

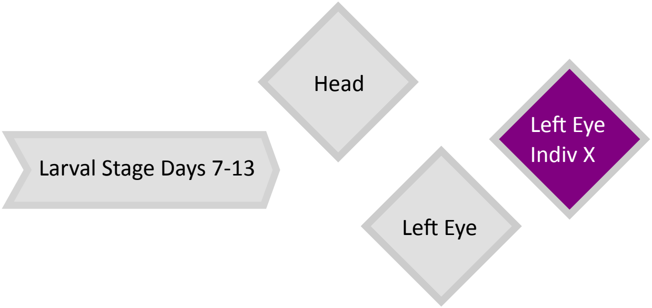
```
[] a rvl:ValueMapping ;
   rvl:sourceValue ro:Requirement ;
   rvl:targetValue common-shapes:Square .
```

We return to value mappings in the next sections, where we specify the calculation of target values from source values (Sect. 7.3) and describe how exactly source and target values can be addressed in RVL (Sect. 7.5).

7.2.5 Inheriting RVL Settings

Setting rvl:inheritedBy allows for extending the results of a defined mapping to related resources. The property defined by rvl:inheritedBy states how other resources need to be related to the originally mapped resource in order to inherit its mapping settings. An example is given below as a listing and a corresponding sketch.

[4] Value Mappings are only allowed when mapping to graphic attributes, not to other graphic relations.



Different classes of the *Zebra Fish Ontology* are distinguished by using a rhombic shape for subclasses of zfo:Fish_Part versus an arrow-like shape for subclasses of zfo:Stage. The mapping to a rhomb is applied to the classes Head and Left_Eye, which are subclasses of Fish_Part but also to the individual Left_Eye_Indiv_X, which is an instance of Left_Eye, resulting in all three objects having a rhombic shape. This is because the shape-mapping is »rvl:inheritedBy rdf:type« to individuals. To distinguish classes from individuals, an additional mapping of rdfs:type to colour is used (not shown in the listing below).

```
zfo:Fish_Part a rdfs:Class ;
zfo:Head rdfs:subClassOf zfo:Fish_Part ;
zfo:Left_Eye rdfs:subClassOf zfo:Head ;
zfo:Left_Eye_Indiv_X rdf:type zfo:Head ;
zfo:Stage a rdfs:Class ;
zfo:Larval_Stage_7_13 rdfs:subClassOf zfo:Stage ;

[] a rvl:PropertyMapping ;
   rvl:sourceProperty rdfs:subClassOf ;
   rvl:targetAttribute viso-graphic:shape_named ;
   rvl:inheritedBy rdf:type ;
   rvl:valueMapping
     [ rvl:sourceValue zfo:Fish_Part ;
       rvl:targetValue common-shapes:Rhomb
     ] ,
     [ rvl:sourceValue zfo:Stage ;
       rvl:targetValue common-shapes:Arrow
     ] .

# (classes of the ZFO renamed and extended for simplicity)
```

For cases, where the defined mapping shall not be inherited, but *exclusively* be used for the related resources, the property rvl:passedTo can be used. For the above example, this would mean that only the instances of Fish_Part will be shown as a rhomb, but not the classes. Besides inheriting via rdf:type, inheritance can also be defined to happen via rdfs:subClassOf or *any* other rdf:Property, including rdfs:subClassOf, to extend the mapping to subclasses, but also dct:isPartOf. Further, mapping settings may also be extended to T-Box relations, using rvl:inheritedBy in combination with rvl:tBoxDomainRange, rvl:tBoxRestriction, or a specific T-Box relation such as owl:someValuesFrom, owl:allValuesFrom or owl:hasValue.

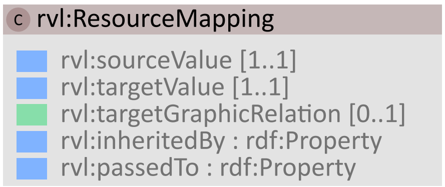
7.2.6 Resource Mapping

rvl:ResourceMappings are convenience mappings for mapping a specific resource to a specific graphic value. Each Resource Mapping may be replaced by a Property Mapping to rdf:ID, additional selector settings and a Value Mapping. However, since mapping resources by their ID is very common (in our case study examples), we introduced a new mapping type for this purpose. Defining Resource Mappings can be the more compact alternative, if only a few values need to be mapped.

The following compact listing shows a resource mapping that maps all resources typed Fish_Part to the shape common-shapes:Rhomb. Since the target value can be unambiguously associated with the graphic relation viso-graphic:shape_named, we can even ommit defining an rvl:targetGraphicRelation:

```
[] a rvl:ResourceMapping ;
   rvl:sourceValue zfo:Fish_Part ;
   rvl:targetValue common-shapes:Rhomb ;
   rvl:passTo rdf:type .

# (classes of the ZFO renamed and extended for simplicity)
```

In general, as for the example of Resource Mappings, many of the mapping types described could be replaced by a mapping that uses a complex pattern matching expression given as a FSL- or SPARQL-selector. While this is flexible and used in other tools such as RDFScape [Spl08], we argue that this may quickly become too complex, when filter expressions become long. Readability improves by introducing extra mapping types for frequently used cases. Still, we aim at minimising the number of mapping types. Therefore, in other cases, we use extra properties as »parameters« instead of extra mapping types. One example for this is the distinction between the mapping of classes and the mapping of instances. Introducing the more general concept of a ResourceMapping (instead of two new types such as »InstanceMapping« and »ClassMapping«), we save one additional mapping type and instead allow for the distinction via the property rvl:inheritedBy. As described in the previous subsection, this allows us to extend the Resource Mapping to instances of the resource's type (rvl:inheritedBy rdf:type) or to its subclasses (rvl:inheritedBy rdfs:subClassOf). A further reduction of the number of mapping types can be achieved via mapping composition (Sect. 7.10).



c rvl:ResourceMapping

🟦 rvl:sourceValue [1..1]
🟦 rvl:targetValue [1..1]
🟩 rvl:targetGraphicRelation [0..1]
🟦 rvl:inheritedBy : rdf:Property
🟦 rvl:passedTo : rdf:Property

7.2.7 Simplifications

Simplifications are not part of the actual visual mapping, but sometimes a necessary preceding step. The intention of a simplification is to gain a simpler model by summarising similar data or removing redundant information. One of the simplifications that can be defined with RVL is rvl:RemoveTransitiveHull, which prevents unnecessary (implicitly given) statements from being displayed *(VC-6)*:

```
[] a rvl:RemoveTransitiveHull ;
   rvl:simplify ex:partOf .
```

Another simplification is rvl:UnifyWithInverse, which selects two inverse properties and defines one of them as the preferred property. The other will be ignored. While inferring inverse statements based on owl:inverseProperty can be handled by OWL reasoners, without this setting, both ex:partOf and ex:hasPart would be displayed redundantly.

```
[] a rvl:UnifyWithInverse ;
   rvl:simplify ex:partOf ;
   rvl:prefer ex:hasPart .
```

7.3 Calculating Value Mappings

A special case of a Value Mapping is a mapping that requires no calculation and maps one or more (domain) source values to exactly one target (graphic) value such as the simple value mapping we already introduced above:

```
[] rvl:sourceValue ro:Requirement ;
   rvl:targetValue common-shapes:Square .
```

This behaviour is similar to what can be achieved by »styling«, and we refer to this as *manual value mapping*. However, the more common case is the *calculated* mapping of sets or intervals of source values to sets or intervals of target values. Fig. 7.5 provides an overview of the ValueMapping class and its attributes. An example of a calculated Value Mapping between two intervals is given below:

```
[] a rvl:ValueMapping ;
   rvl:sourceValueInterval [
     rvl:lowerBoundIncl "0" ;
     rvl:upperBoundIncl "50"
   ] ;
   rvl:targetValueInterval [
     rvl:lowerBoundIncl "0" ;
     rvl:upperBoundIncl "100"
   ] .
```

It will frequently happen that source values from the domain data and the target graphic values have a different scale of measurement, or, they have the same scale, but the source values are continuous and the target values are discrete (or the other way round). Sometimes, also a discretisation of values will be required while calculating the mapping. In this section, we give an overview of these more complicated cases. Fig. 7.6 shows a matrix of three main scales of measurement that we distinguish (*nominal, ordinal and quantitative*) for both source and target values. For a detailed description of the semantics of all cases refer to the appendix or the RVL specification provided at http://purl.org/rvl/ValueMapping. Below, we give a brief tabular overview of the ten cases that are sketched schematically in Figure 7.6:

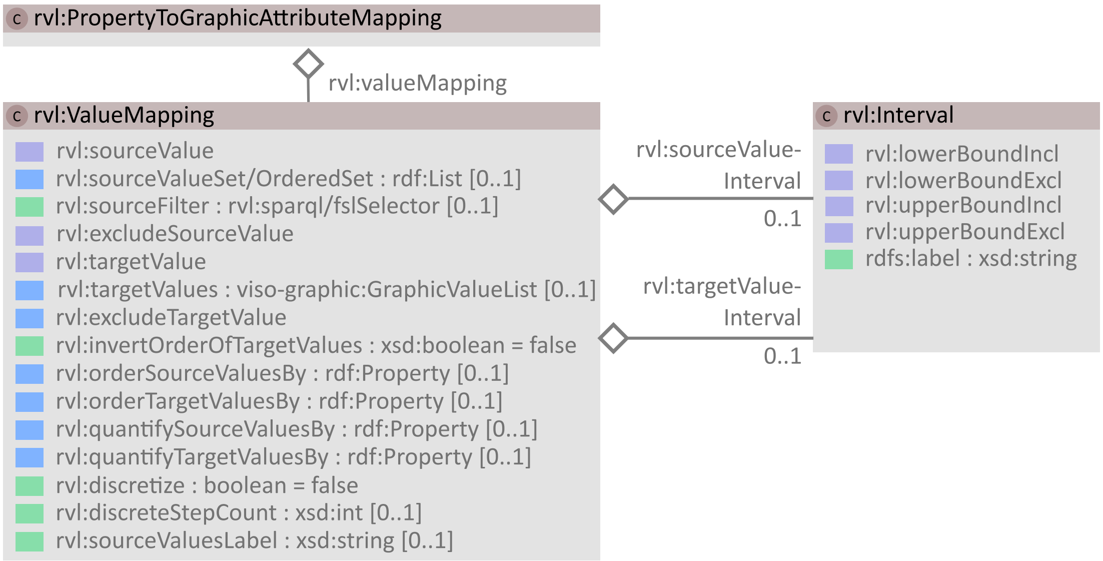

rvl:PropertyToGraphicAttributeMapping
rvl:valueMapping
rvl:ValueMapping
rvl:sourceValue
rvl:sourceValueSet/OrderedSet : rdf:List [0..1]
rvl:sourceFilter : rvl:sparql/fslSelector [0..1]
rvl:excludeSourceValue
rvl:targetValue
rvl:targetValues : viso-graphic:GraphicValueList [0..1]
rvl:excludeTargetValue
rvl:invertOrderOfTargetValues : xsd:boolean = false
rvl:orderSourceValuesBy : rdf:Property [0..1]
rvl:orderTargetValuesBy : rdf:Property [0..1]
rvl:quantifySourceValuesBy : rdf:Property [0..1]
rvl:quantifyTargetValuesBy : rdf:Property [0..1]
rvl:discretize : boolean = false
rvl:discreteStepCount : xsd:int [0..1]
rvl:sourceValuesLabel : xsd:string [0..1]
rvl:sourceValue-Interval
0..1
rvl:targetValue-Interval
0..1
rvl:Interval
rvl:lowerBoundIncl
rvl:lowerBoundExcl
rvl:upperBoundIncl
rvl:upperBoundExcl
rdfs:label : xsd:string

Figure 7.5: Overview of the rvl:ValueMapping class.

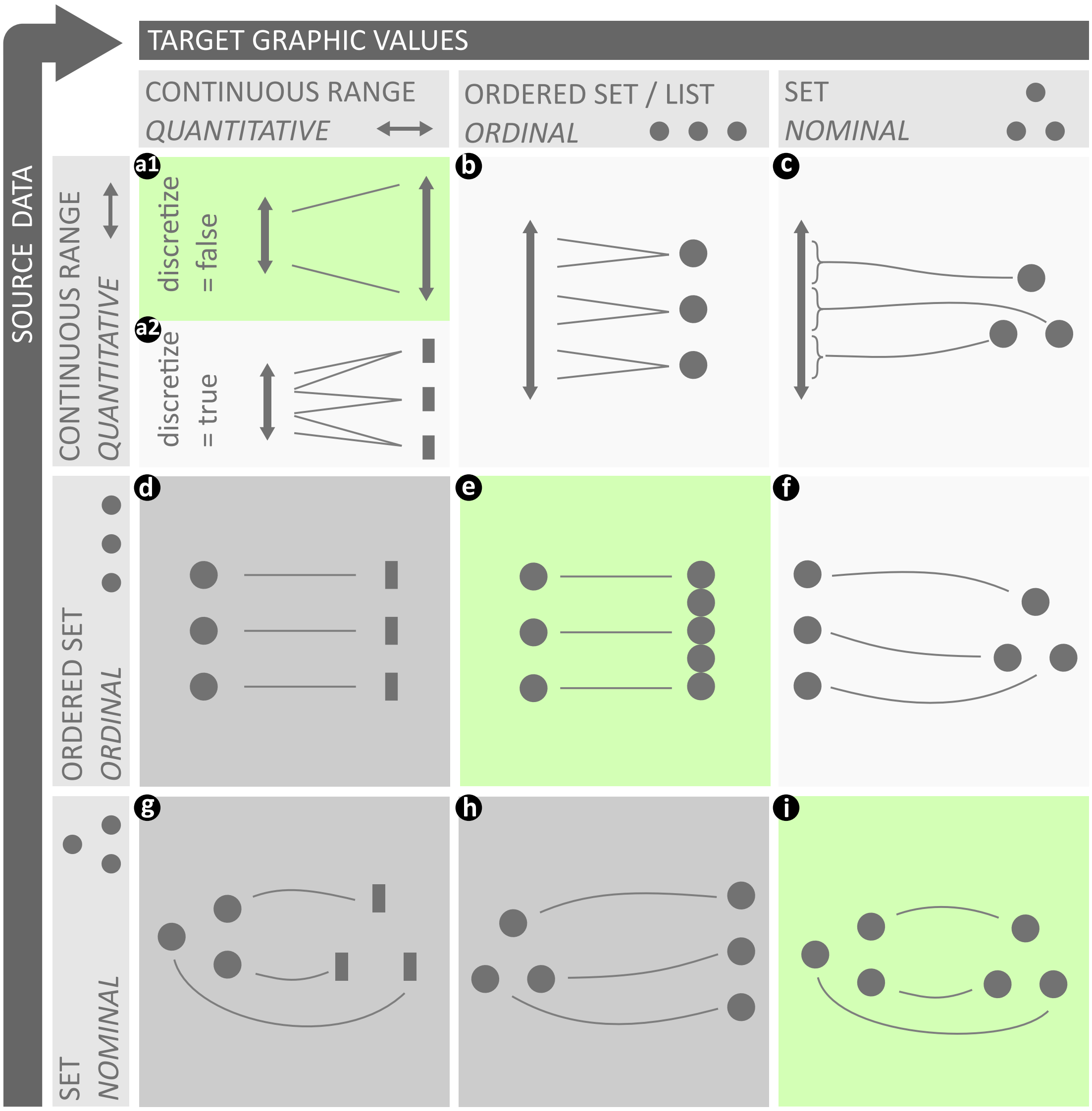


Figure 7.6: Calculation of value mappings depending on the scale of measurement of both source data and graphic values taking part in a mapping. For each cell of this matrix, we state by its background colour whether this mapping case leads to information loss (white; a2, b, c, f), the target graphic attribute is not fully exploited with respect to its ability to express the scale of measurement (grey; d, g, h) or whether the scale of source and target values exactly match (green; a1, e, i).

a1	continuous range ↦ continuous range	**If discretize = false** Map the maximum range of the available source values to the maximum range of available target values.
a2	continuous range ↦ continuous range	**If discretize = true & discreteStepCount not set** Discretize the range of available target values into x steps, where x is the number of named values in the target range. **If discretize = true & discreteStepCount set** Discretize the range of available target values into rvl:discreteStepCount steps.

Case *a1* is the most simple case of a mapping between continuous source and target values, both scales being quantitative. For each source value a concrete target value can simply be calculated by scaling the source to the target range. (Recall that if no intervals are explicitly defined, these ranges are calculated as the span from the lowest to the highest value). Case *a2* is similar to *a1*, but with the rvl:discretize flag being set. Furthermore, we distinguish, whether a rvl:discreteStepCount is defined, i. e., whether the mapping explicitly defines into how many discrete steps the range of target values shall be split. If rvl:discreteStepCount is not set, RVL interpreters should use the default set (Sect. 7.7) of discrete (named) values that corresponds to the chosen graphic attribute.

b	continuous range ↦ ordered set	Discretise the source values into x steps, where x is the number of values in the target set. Then map all source values belonging to the first step to the first target value, those belonging to the second step to the second target value ...[5]
c	continuous range ↦ set	Discretise the source values into x steps, where x is the number of values in the target set. Then map all source values belonging to the same step to some (but different) target value.[6]

In case b, the target values are already discrete (an ordered set or list of named values), so only the source values have to be discretised. The number of discretisation steps is determined by the size of the ordered set (respectively the length of the list). While in case b, the source and target values are matched with respect to their order, for case c, the target values are unordered. Therefore, every source value is simply mapped to one of the remaining target values[7].

[5] discreteStepCount could also be allowed here.

[6] See footnote 5.

[7] Ideally, it should be avoided that a subset of values is accidentally selected that appears to be ordered (e. g., the named colours *orange*, *red*, *light-orange*).

d, g	(ordered) set ↦ continuous range	Discretise the range of available target values into x steps, where x is the number of source values. **If discreteStepCount set** Discretise the range of available target values into discreteStepCount steps.
e	ordered set ↦ ordered set/list	**If $\|$source set$\| \leq \|$target set$\|$** Stretch the available range of ordered target values to the whole range of source values (similar to mapping intervals of continuous data). **If $\|$source set$\| > \|$target set$\|$** Issue a warning and ask if cycling should be used (will be ambiguous except in special cases, where the context helps to distinguish the values).
f, h, i	(ordered) set ↦ (ordered) set/list	**If $\|$target set$\| = 1$** Use the same target value for all source values. **Else if $\|$source set$\| \leq \|$target set$\|$** Use a random subset of target values. **Else if $\|$source set$\| > \|$target set$\|$** Issue an error.

For each mapping case shown in Fig. 7.6, we indicate by the background colour whether the mapping case leads to information loss (white), the target graphic attribute is not fully exploited with respect to its ability to express the scale (grey) or whether the scale of source and target values exactly match (green)[8]. Information may also be lost when discretisation happens during the calculation, e. g., in case *a2*, we have to accept some information loss, which may be tolerable for the sake of achieving clear distinct values that can more easily be distinguished.

7.4 Defining Scale of Measurement

When calculating Value Mappings, RVL engines must act differently depending on the scale of measurement of source data and target graphic attributes, as we described in the last section. In the following, we introduce the options that the RVL and VISO vocabularies offer for specifying these scales. Furthermore, we describe how RVL engines should proceed in order to derive scales from explicit settings, implicit settings and the data itself.

For *globally* defining the scale of measurement of a property, vocabulary from the *VISO/data* module can be used. Making ex:size a subproperty of viso-data:has_quantitative_value tells RVL engines that values of the property size are *quantitative*, for instance:

```
ex:size rdfs:subPropertyOf viso-data:has_quantitative_value .
```

[8] Although this is similar to the definition the two expressiveness criteria by Mackinlay [Mac86a], we do not consider the expressiveness of graphic relations directly in the RVL specification, but, here, we only refer to the relation between the two involved scale of measurements (the scale of the source and the scale of the target values). Statements, as defined by Mackinlay, such as »saturation can only express ordinal and quantitative values« are not considered at the language specification level, but should be loaded from a *VISO/facts*-based knowledge base by visualisation systems.

In some cases, setting the scale of measurement globally may not be desired, but needs to be done *locally*, for the context of a Value Mapping. In other cases, global settings may not be possible, e. g., if no numeric values at all are attached to the resources. Multiple options exist for locally defining the scale of measurement. The first is to define a property to order or quantify the source values, e. g.:

```
ex:PersonA ex:receivedAward ex:Silver ;

ex:Bronce ex:ranked "3"^^xsd:float ;
ex:Silver ex:ranked "2"^^xsd:float ;
ex:Gold ex:ranked "1"^^xsd:float ;

[] a rvl:ValueMapping ;
  rvl:orderSourceValuesBy ex:ranked ;

      ... other settings of the Value Mapping ...
```

Here, rvl:orderSourceValuesBy is used to select the relation ex:ranked for ordering the resources *Bronce, Gold, Silver*. This allows for ranking these resources differently in different situations. Setting rvl:orderSourceValuesBy resp. rvl:quantifySourceValuesBy can be omitted, if values are assigned via *VISO/data* properties such as viso-data:has_quantitative_value, as these properties will be evaluated by default. For assigning order, two other options exist: First, an (RDF) list of the resources may be used. Using a list allows for selecting certain values while omitting others. Lists may be created ad hoc and anonymously in a mapping or given a name for reuse.

```
ex:CityA ex:size ex:Medium ;
ex:CityB ex:size ex:Big ;

[] a rvl:ValueMapping ;
  rvl:sourceValueOrderedSet (
        ex:Big
        ex:Medium
        ex:Small
        ) ;

      ... other settings of the Value Mapping ...
```

Second, a property relating the various resources may be selected. In the example below, the scale of measurement for ex:size is given by defining the property ex:gt (greaterThan) as an order relation. This way, the order of the size values *Small, Medium, Big* is defined.

```
ex:Big ex:gt ex:Medium .
ex:Medium ex:gt ex:Small .

[] a rvl:ValueMapping ;
  rvl:orderSourceValuesBy ex:gt ;

      ... other settings of the Value Mapping ...
```

7.4.1 Determining the Scale of Measurement

The decision diagram in Fig. 7.7 describes how to calculate the scale of measurement of source values from statements defined with the RVL and VISO-data vocabularies. The scale of measurement can be derived from explicitly and implicitly stated global or local settings or – if no other information is available – also be guessed from the source values' data type. Advanced guessing of the scale of measurement is beyond the scope of this work. However, we specify two domain-agnostic cases based on the type of source literals that are used: For xsd:int, we assume an ordinal scale of measurement and for xsd:float, we assume a quantitative scale of measurement. In all remaining cases, we can only assume a nominal scale.

The calculation of the scale of measurement for the *target* values (not shown here) could be done analogous to the calculation of the scale of measurement for the source values in Fig. 7.7, but, since we require every graphic relation to explicitly define its scale of measurement, no such calculation is necessary.

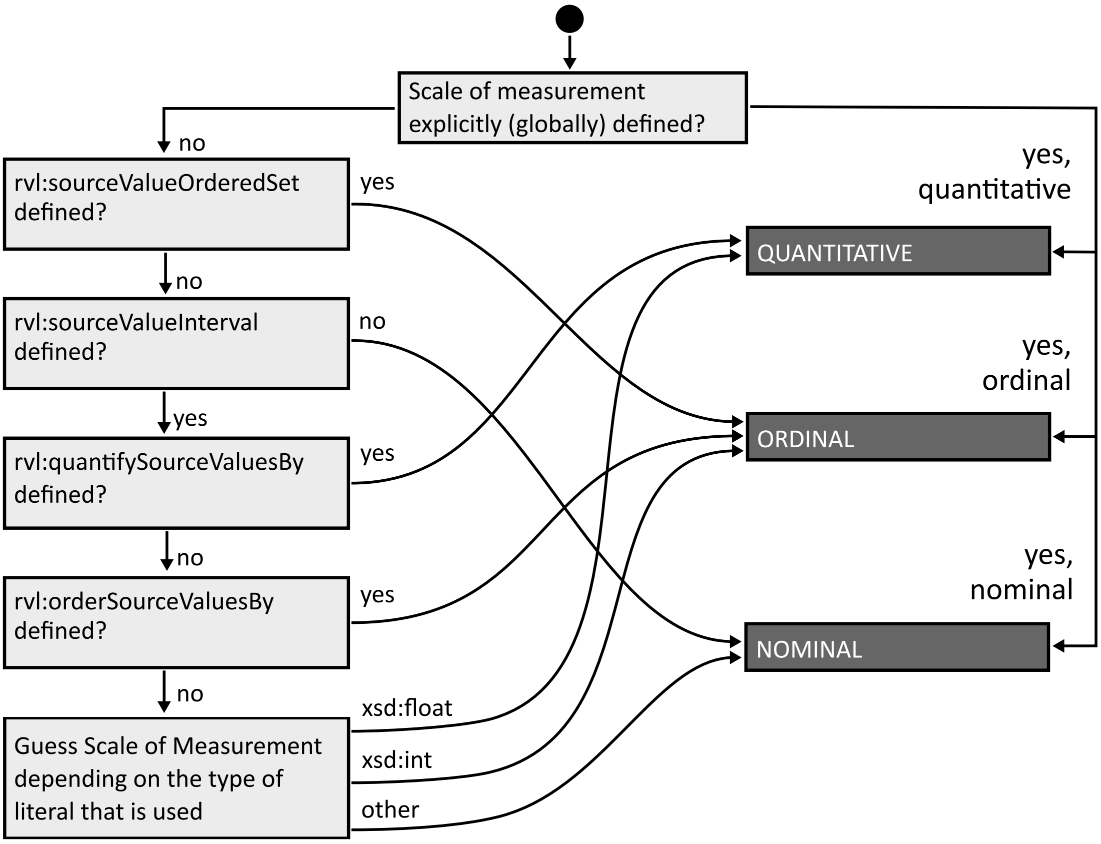

Scale of measurement explicitly (globally) defined?
no
yes, quantitative
yes, ordinal
yes, nominal
rvl:sourceValueOrderedSet defined?
yes
no
rvl:sourceValueInterval defined?
no
yes
rvl:quantifySourceValuesBy defined?
yes
no
rvl:orderSourceValuesBy defined?
yes
no
Guess Scale of Measurement depending on the type of literal that is used
xsd:float
xsd:int
other
QUANTITATIVE
ORDINAL
NOMINAL

Figure 7.7: Decision diagram specifying how to determine the scale of measurement of source values.

7.5 Addressing Values in Value Mappings

In addition to the scale of measurement, the exact set of source and target values is required for the calculation of value mappings. Therefore, in this section, we introduce the value selectors that RVL offers and specify how to determine the set of source and target values from the possible selector settings.

RVL defines various selectors to allow for selecting intervals (ranges) of values by specifying a minimum and/or maximum value as well as for selecting ordered and unordered sets of values. Besides selecting values to be *included*, single values may also be *excluded*. If even more complex selectors are required, again filter expressions, e. g., in SPARQL, can be used. In Fig. 7.5, we already gave a first overview of these selectors. Selector-properties exist for both source and target side of a value mapping. We first look at the selection of *source* (domain) values and then turn to the *target* (graphic) values.

7.5.1 Determining the Set of Addressed Source Values

Value Mappings can address one or multiple source values. In the following, we specify how to derive these values from the various possible source value selectors that can be used with a Value Mapping. Fig. 7.8 shows this process by means of a decision diagram. In a first step, it needs to be distinguished whether the Value Mapping requires a calculation of values at all. This is obviously not given, if only a single source value is selected using rvl:sourceValue – in this case,

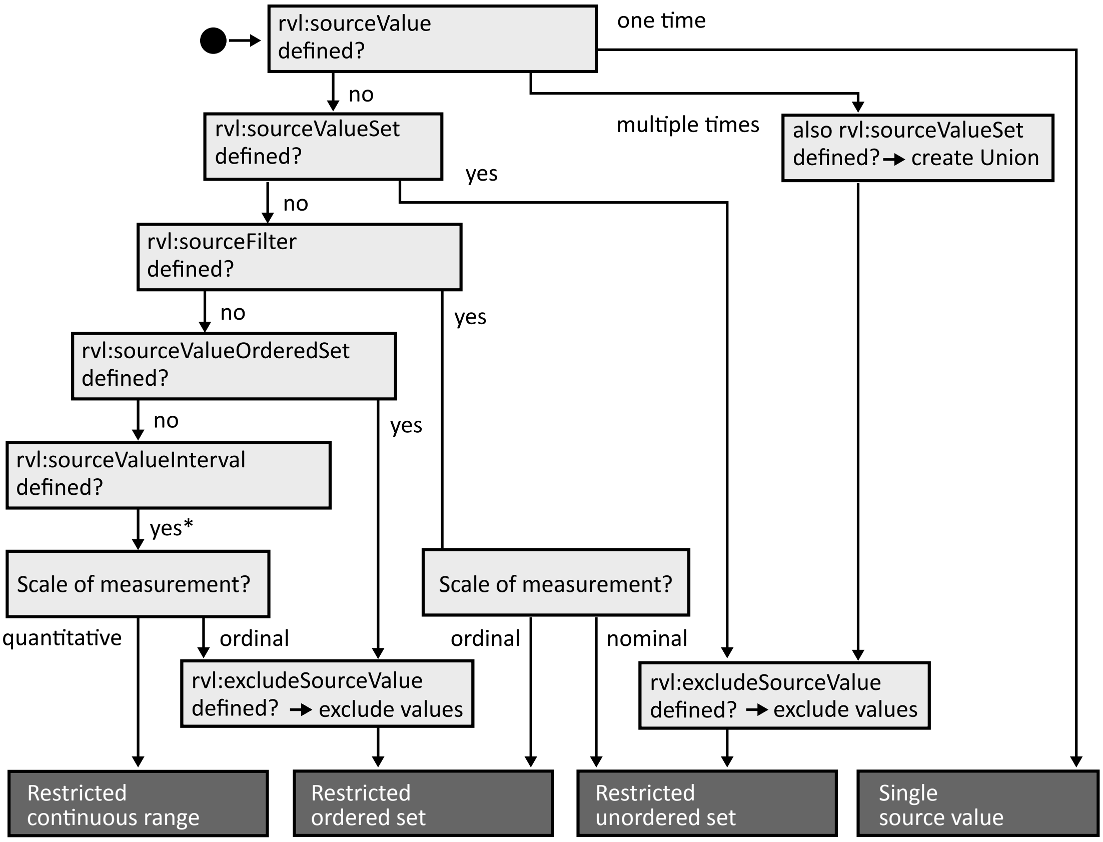

rvl:sourceValue defined?
one time
no
multiple times
also rvl:sourceValueSet defined? → create Union
rvl:sourceValueSet defined?
yes
no
rvl:sourceFilter defined?
no
yes
rvl:sourceValueOrderedSet defined?
no
yes
rvl:sourceValueInterval defined?
yes*
Scale of measurement?
Scale of measurement?
quantitative
ordinal
ordinal
nominal
rvl:excludeSourceValue defined? → exclude values
rvl:excludeSourceValue defined? → exclude values
Restricted continuous range
Restricted ordered set
Restricted unordered set
Single source value

Figure 7.8: Decision diagram specifying how to determine the set of source values addressed by a value mapping.
*) If no source values are defined by any means this leads to an invalid mapping. This case and other combinations of parameters leading to invalid mappings are not shown in the diagram.

we simply map the source value to the specified target value (*manual value mapping*). Manual value mappings are only allowed when a single target value is selected. Depending on whether other selector properties such as rvl:sourceFilter or rvl:sourceValueInterval are set and also taking into consideration the scale of measurement, which we identified beforehand, we decide on a restricted range, respectively a restricted ordered or unordered set of values. Sets of restricted source values can further be reduced by excluding single values using rvl:excludeSourceValue. While most of the selectors are to be used alternatively, those selecting an (unordered) set of values can be used in combination. The union of all selected values will be used in this case.

7.5.2 Determining the Set of Addressed Target Values

Determining the addressed target values is similar to the calculation of source values, but still differs in a few points. As for the source values, the calculated mapping of values only takes place, if more than a single target value is selected. In the case of a single target value, all source values simply receive the same target value (manual value mapping). Since less selector properties exist for the target side, fewer decisions are required. Only for value ranges (rvl:targetValueInterval)

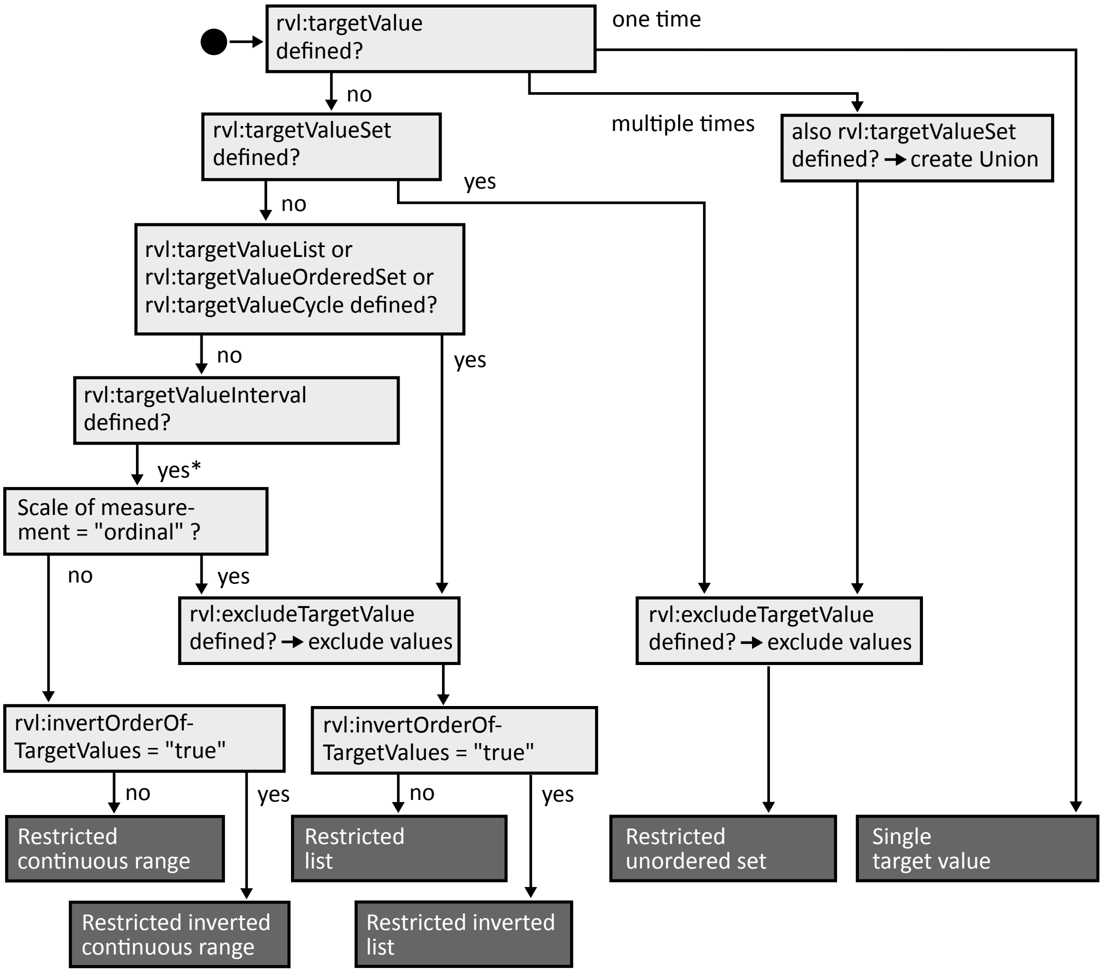

rvl:targetValue defined?
one time
no
multiple times
rvl:targetValueSet defined?
also rvl:targetValueSet defined? → create Union
yes
no
rvl:targetValueList or rvl:targetValueOrderedSet or rvl:targetValueCycle defined?
no
yes
rvl:targetValueInterval defined?
yes*
Scale of measure-ment = "ordinal" ?
no
yes
rvl:excludeTargetValue defined? → exclude values
rvl:excludeTargetValue defined? → exclude values
rvl:invertOrderOf-TargetValues = "true"
rvl:invertOrderOf-TargetValues = "true"
no
yes
no
yes
Restricted continuous range
Restricted list
Restricted unordered set
Single target value
Restricted inverted continuous range
Restricted inverted list

Figure 7.9: Decision diagram specifying how to determine the set of target values addressed by a value mapping.
*) If no target values are defined by any means this leads to an invalid mapping. This case and other combinations of parameters leading to invalid mappings are not shown in the diagram.

we need to distinguish ordinal from quantitative ranges. However, as opposed to source values, there is an additional property rvl:invertTargetValues. This allows, for example, for reusing an existing list of target values that fits the intended visualisation purpose, except having the wrong orientation. As for source values existing unordered sets of target values can be extended »on the fly« by additional values, since the union of all value sets will be processed, and single values can be excluded.

7.6 Overlapping Value Mappings

If a source value is mapped by multiple Value Mappings, the priority of the Value Mapping – an integer between 1 (highest priority) and 5 (lowest priority) – is used to determine the mapping to be applied. Additionally, a warning should be issued. The highest priority have Value Mappings of single source values, followed by ordered sets, lists and, finally, sets and intervals having the lowest priority:

Selectors used in the value mapping	Priority	Examples
Single value	1	`rvl:sourceValue ex:Dresden ;`

Ordered set / list	2	`rvl:sourceValueOrderedSet`
		`(ex:Small ex:Medium ex:Large) ;`

Ad hoc set	3	`rvl:sourceValueSet`
		`(ex:Dresden ex:Berlin ex:Munich) ;`

Named set	4	`rvl:sourceValueSet ex:ExampleSetOfCities ;`

Interval	5	`rvl:sourceValueInterval [`
		`rvl:lowerBoundIncl "0" ;`
		`rvl:upperBoundIncl "50" ] ;`

This means that in the following example, ex:Dresden will be mapped to viso-graphic:Green, since the single selected value has a higher priority (1) than the ad hoc created set of values (3): ...

```
ex:SomeValueMapping a rvl:ValueMapping ;
  rvl:sourceValueSet (ex:Dresden ex:Berlin ex:Munich) ;
  rvl:targetValue viso-graphic:Red .

ex:AnotherValueMapping a rvl:ValueMapping ;
  rvl:sourceValue ex:Dresden ;
  rvl:targetValue viso-graphic:Green .
```

7.7 Default Value Mapping

As initially mentioned, property mappings can also be created without adding an explicit value mapping. Assuming reasonable defaults *(VC-12)* yields very compact mappings:

```
ex:SimplePM a rvl:PropertyMapping ;
  rvl:sourceProperty rdf:type ;
  rvl:targetGraphicRelation viso-graphic:color_named .
```

A default set or range of graphic attributes can be defined with rvl:defaultValueSet respectively rvl:defaultValueRange. In the following example, we define a default set of target values for the graphic attribute *colour (named)*. We choose a predefined set of colours from our example facts collection *empiric-facts*:

```
viso-graphic:color_named rvl:defaultValueSet empiric-facts:ColorBlindSafeColorSet .
```

Similarly, for the attribute *x-position*, we could define that – by default – all mappings to x-position will cover a range from 0–100 (percentage of the available screen space):

```
viso-graphic:x_position rvl:defaultValueRange [
  a rvl:Interval ;
  rvl:lowerBoundIncl "0" ;
  rvl:upperBoundIncl "100" ] .
```

Following the convention over configuration principle, we might even omit the previous setting, since whenever no default value range is defined with RVL, the maximum range defined in VISO/graphic is used. For graphic attributes such as viso-graphic:shape_named, a trivial default value set can be retrieved by selecting all graphic attribute values (defined as named resources like viso-graphic:Square) that are available to the RVL interpreter. Once a default target value range or set is identified, a default value mapping can be performed by mapping all values available in the source data to the default value range or set.

7.8 Default Labelling

Labelling graphic objects is done by default. The labelling is realised by superimposing the label over the labelled object. The text value of the label is derived from the rdfs:label of the represented resource in first place, followed by the local name of the resource if not available. Defining alternative text values could be passed to Fresnel in future versions of RVL. The specification currently lacks a means to prevent the automatic labelling. One option could be to shift all default settings to a basic set of (default) RVL mappings that can be disabled on demand.

7.9 Defining Interaction

The current RVL specification allows for defining basic interactions. No specific RVL constructs are provided for interactions; rather general Property Mappings can be used in combination with special target graphic relations from the VISO. The following example defines that the ex:borders relation should result in a viso-graphic:co-highlight relationship between the graphic objects representing the bordering countries. That means all graphic objects taking part in the borders relation should be highlighted, whenever one of them is »activated«.

```
[] a rvl:PropertyMapping ;
   rvl:sourceProperty ex:borders ;
   rvl:targetGraphicAttribute viso-graphic:co-highlight .
```

While this example shows that some interactions can be concisely defined using Property mappings, it also raises the question how similar, slightly more complex tasks should be solved. For example, we might want to limit co-highlighting to one direction for non-symmetric properties. Also, we did not specify when the highlighting should be triggered – on hovering or on clicking? Probably, new RVL mapping constructs are required to specify such interactions. The specification should also be aligned with the corresponding selectors in CSS.

Another observation is that for some interactions, we do not need to reference relations in the source data, but relations from the AVM. This is, for instance, the case for co-highlighting graphic objects that have been duplicated to occur at various places in a tree. Allowing for AVM-relations such as avm:representsSameResourceAs[9] as rvl:sourceProperty could possibly describe such scenarios in a consistent manner.

[9] avm:representsSameResourceAs may be derived from avm:represents relations.

7.10 Mapping Composition and Submappings

We distinguish two basic types of mapping compositions that can currently be described with RVL – *simultaneous composition* and *context composition*. Although many of the sketches from the case studies can be implemented with these compositions *(VC-7)*, further work on composition is required based on the composition cases we list in Sect. 8.2.1.

Mappings taking part in a **simultaneous composition** can be added to or removed from a set of mappings[10] independently from one another. Each mapping works on the same graphic objects without depending on the other mappings. Mappings taking part in a **context composition** cannot be changed independently of one another, because one mapping needs to work on graphic objects that are identified by a role assigned by the other mapping. Sometimes it will even work on objects that are newly created by the other mapping.

In RVL, a second mapping can be attached to a first one via an rvl:SubMappingRelation. The graphic object to work on (graphic context) is identified by its role using the property rvl:submapping_onRole *(VC-9)*. The data that the submapping should process (data context) is defined by pointing to either subject, predicate or object of the first mapping using rvl:submapping_onTriplePart[11]. To clarify this, in Fig. 7.10, we give another complete example of an RVL mapping (process). The example refers to the sketch from Fig. 3.4c, depicting the visualisation of documents and their citation relations. While we used a UML-like notation to represent the involved RVL mappings for the use case described in Fig. 7.10, the same mapping in Turtle notation is given below. Although the Turtle syntax is more human-readable than the RDF/XML notation, it becomes clear that for building more complex composed mappings, tooling is required:

```
[] a rvl:PropertyMapping ;
  rvl:sourceProperty cito:cites ;
  rvl:targetGraphicRelation viso-graphic:Linking_Directed_Relation ;
  rvl:hasSubMapping [
    rvl:submapping_onRole viso-graphic:linking_connector ;
    rvl:submapping_onTriplePart rvl:predicate ;
    rvl:submapping_mapping [
      a rvl:PropertyMapping ;
      rvl:sourceProperty rdf:ID ;
      rvl:targetGraphicAttribute viso-graphic:color_named
    ]
  ] .
```

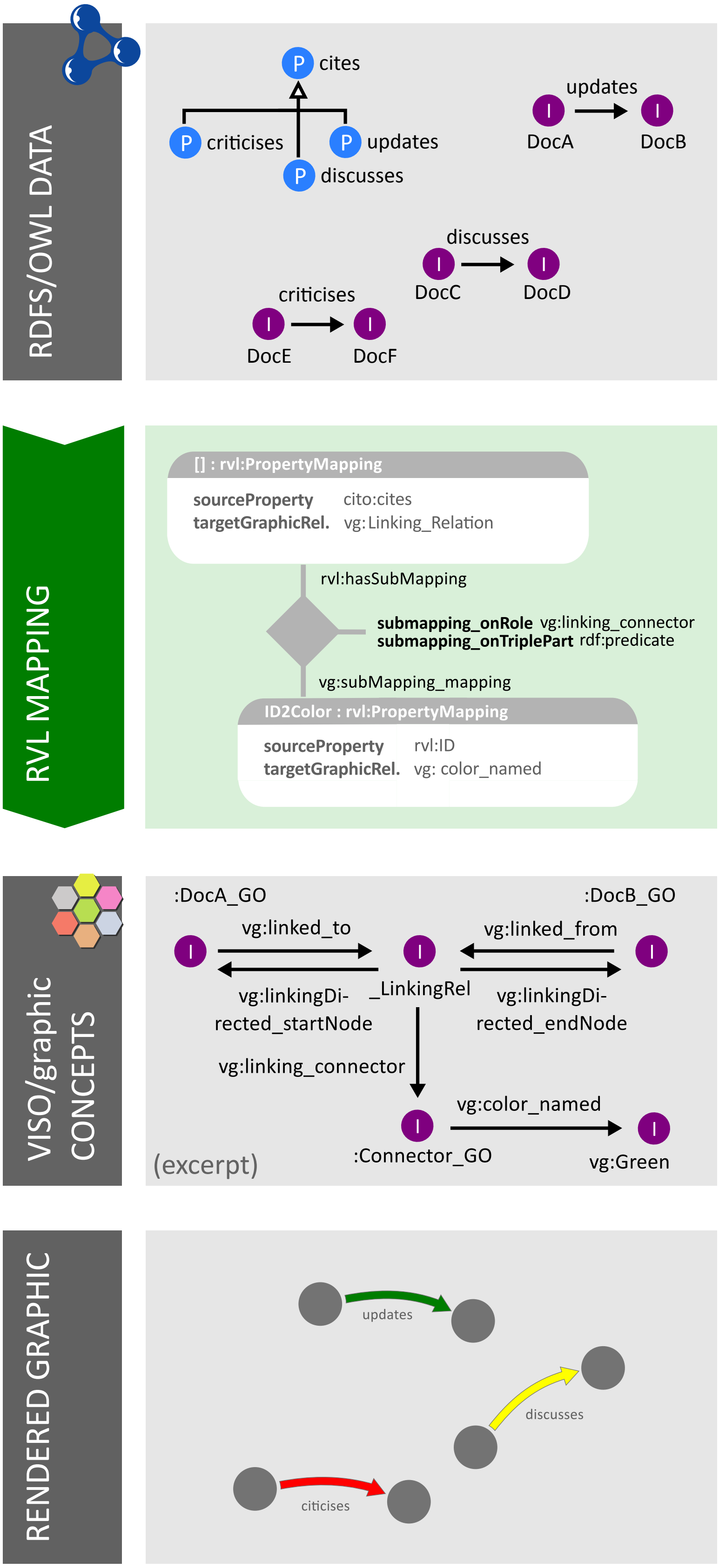
7.11 A Schema Language for RVL

This section is about the schema of RVL and discusses languages that could be used to define this schema. Since it is not about the features of RVL itself, readers not interested in the schema, may continue with Sect. 7.12.

To clarify the distinction between mappings defined with RVL, the RVL schema, and languages used to define this schema, Table 7.1 compares the different »language levels« in the context of RVL. We first look at the latter, i. e., languages that can be used to define schemata. For defining the schema of RVL, multiple options exist, which we discuss in the following, based on three additional requirements that we put with respect to the schema. At the end of this section, we briefly describe our approach for defining the RVL schema using SPIN (introduced in Sect. 2.2.7) constraints.

[10] Currently, there is no such construct as a *mapping set* defined in the RVL specification, hence the set of all (enabled) mappings defined in the RVL mapping model is used in our prototypes. However, this seems to be an obvious future extension of RVL to allow for switching between mapping sets, i. e., switching between multiple graphics. Alternatively, one named RDF graph per graphic could be used.

[11] Defining rvl:submapping_onTriplePart is optional. By default, the subject will be used.



(1) The source data that we want to visualise consists of three pairs of documents, where one document is related to the other via a different subproperty of the cito:cites relation, namely cito:updates, cito:discusses and cito:criticizes.

(2) The visual mapping is realised by two rvl:PropertyMappings – one for mapping cito:cites to linking and another to distinguish the exact kind of citation by colour. The context composition of the two mappings is defined by attaching the second one as a submapping to the graphic object with the role vg:linking_connector. Further, we define that the submapping should be based on the rdf:predicate of the statement handled by the first mapping.

(3) The result is an abstract visual model of a graphic (AVM, Chapter 6), consisting of graphic objects and relations. For each document, a corresponding graphic object has been created (suffixed »_GO«) and graphically related to the cited document via a vg:Linking_Directed_Rel.. Each connector is assigned a different attribute value for vg:color_named, using the default set of colours.

(4) The rendering of the final graphic is not defined by RVL but only shown here for illustration purposes. The rendering engine picked arrows to implement the directed connectors and layouted the nodes. Concrete RGB values have been looked up and assigned to the arrows during rendering.

Figure 7.10: Example of an RVL mapping using submappings.

Language level	Name	Examples
Meta-languages & Schema languages	RDF(S), OWL, SPIN	rdfs:Class is an rdfs:Class.
Language & Schema	RVL defined by the RVL Schema	rvl:PropertyMapping is an owl:Class. rvl:PropertyMapping has exactly 1 rvl:targetGraphicRelation.
Model	Concrete RVL mappings	ex:Cost2LightnessMapping is an rvl:PropertyMapping.

Table 7.1: Overview of language levels in the context of RVL (examples are given in natural language).

LR-15 The schema of the language must be restrictive and expressive enough to derive tooling from it.

LR-16 The schema language must be aware of ontology semantics, not only URIs.

LR-17 Constraints in the mapping language's schema and constraints defined in *VISO/facts* should be handled consistently

First, in order to allow for the generation of mapping editors from the language description, the RVL schema should define, in a tool-usable way, what is a valid mapping in RVL *(LR-15)*. Changes to the language definition should automatically lead to changes in the editor, which is something that metamodelling enables. If we could derive a guided editor for RVL directly from the language's schema, constraints already specified in the schema would not have to be redundantly hard-coded in the source code of guidance tools. This would contribute to the extensibility of RVL.

Second, since we want to easily define constraints on our mapping types, the schema language should be aware of ontology semantics *(LR-16)*. It will frequently occur that constraints of the mapping language have to reference *VISO/graphic* terms, such as in the following constraint (here defined in natural language):

> »An rvl:PropertyMapping
> always maps an rdf:Property
> to a viso-graphic:GraphicRelation.«

Third, an additional requirement exists, because we want to use RVL in a semi-automatic visualisation system: External rules on graphic syntax and human perception based on the *VISO/facts* module will have to be accessed for constructing editors. Therefore, we require that both the constraints from the mapping language's schema and the constraints defined with *VISO/facts* can be handled consistently *(LR-17)*.

One option was to stay completely within the (RDF-based) ontology technological space. This is suggested by the fact that both the source data and the graphic elements we are mapping onto are RDF-based ontologies. Defining also our mapping language RVL with RDF-based technologies, therefore, could help avoiding technological gaps. Defining restrictions that use terms from the source ontologies and VISO could easily be done, e. g., via OWL class restrictions and also the domain and range of properties could easily be stated with OWL. An additional benefit is that mapping definitions, instantiating an RDF-based vocabulary, could conveniently be shipped along with the data they are visualising. Furthermore, since each mapping was an RDF resource, globally uniquely identified via a URI, linked data principles would apply to it. This would contribute to the requirement of shareable mappings, since other users could dereferentiate mappings and reuse them in their own visualisations. The authors of Fresnel chose this approach and defined the RDF presentation vocabulary on top of OWL (cf. left column of Table 7.2). The problem with this approach is that class restrictions and domain–range settings defined in OWL are not meant to prescribe valid user input, but to derive new knowledge under the Open World Assumption, as we discussed in more detail in Sect. 4.4. Since the regular OWL semantics and the corresponding tools are not applicable, we do not consider OWL (alone) appropriate to define the RVL language. While OWL may also be interpreted with different (closed world) semantics and specific tooling could be built, OWL also lacks constructs such as defaults and attributes for conveniently defining a rich prescriptive schema.

Another option was to write the RVL schema in a different technological space, such as the metamodelling world or the grammar world [PSA$^+$12], and only reference the ontology resources via their URIs (right column of Table 7.2). If the mapping language was defined by grammar rules or metamodel constraints, under a closed world assumption, tooling for constrained-based guidance (editors, warning messages, auto-suggest functions) could conveniently be generated based on these constraints with existing technologies. On top of Ecore and EMF (cf. Sect. 2.2.6), which became a popular base for metamodelling, many frameworks support building textual or graphical editors for Ecore-based languages (Sect. 4.4.1). On the downside, if this means that ontologies need to be transformed, (e. g., to Ecore), it will be difficult to dynamically adapt to extensions of the ontological models. Re-modelling ontologies in Ecore is a drawback, if we need to access facts from external knowledge bases, which may be subject to frequent change.

Our choice for RVL was to select the first option and stay within the (RDF-based) ontology technological space. However, we use RDFS/OWL only for modelling the abstract syntax of RVL, and use SPIN for defining the constraints of the RVL schema (centre column of Table 7.2). With TopBraid Composer[12], a modelling environment is available that can be used to build an editor that supports guidance for creating valid RVL mappings and, at the same time, allows for conveniently accessing *VISO/graphic* resources as well as visualisation facts from a knowledge base such as *VISO/facts/empiric*. Table 7.2 summarises our comparison of the three options described above under the aspects of expressiveness, use of standards, support of shareability of mappings, the availability of tooling and the support of guidance based on both schema knowledge and external facts from existing knowledge bases. In the right column, we use the concrete solution of OWL+OWL-CL (used by OWLText, introduced in Sect. 4.4.3) as an example for the second approach.[13]

7.11.1 Concrete Examples of the RVL Schema defined with RDFS/OWL and SPIN

In the following, we provide a set of concrete examples to illustrate which parts of RVL are defined with RDFS/OWL, respectively SPIN, and how a SPIN constraint can be defined. Types and relations of RVL are defined with RDFS and OWL[14]:

```
rvl:PropertyMapping rdfs:subClassOf rvl:Mapping.

rvl:sourceProperty a rdf:Property.

rvl:Clamp
  a rvl:OutOfBoundHandlingType ;
  rdfs:label "clamp"^^xsd:string ;
  dct:description "Values outside the defined interval are
                   set to the interval’s boundaries."^^xsd:string .
```

[12] http://www.topquadrant.com/topbraid/, accessed: 02.07.2015.

[13] A comparison of SPIN to other rule/constraint languages has been done by Pooran Patel in a master thesis [Pat13], confirming our choice of SPIN.

[14] The usage of OWL is not shown in the example; RVL employs OWL property classes such as owl:ObjectProperty.

	Pure OWL used to describe Fresnel	**OWL+SPIN** our approach for RVL	**OWL+OWL-CL** used within OWLText
Standard reasoning – Open/Closed world			
	OW	OW+CW	CW
Referencing terms of existing ontologies			
	++	++	– (requires remodelling in Ecore)
Expressiveness			
– Defaults	– (natural language)	x	x
– Path selector language	x (FSL)	(x) property paths in SPARQL 1.1	–
Shareability	++	++	+
Use of standards	RDF(S), OWL, SPARQL	RDF(S), OWL, SPARQL	OWL
Non-standards	FSL	SPIN	OWL-CL
Availability of tooling that considers the constraints			
– Textual editor	–	– (textual representation not human readable)	x (EMFtext)
– Graphical editor	–	x (TopBraid Composer; list handling inconvenient)	–
Guidance support based on schema			
– Warnings	+	++	++
– Constrained UI	+	+	+
Guidance support based on existing referenced knowledge bases			
– Warnings	–	++	–
– Constrained UI	–	+	–

Table 7.2: Comparison of three techniques to define schemata and derive tooling from these schemata: The left column represents the approach of staying completely within the ontology technological space, exemplified by the solution chosen to define Fresnel. (FSL stands for the Fresnel Selector Language, cf. Sect. 4.3.1.) The right column represents the approach of bridging the ontology and metamodelling technological space, exemplified by OWLText. The centre column shows our choice of using OWL in combination with SPIN for the definition of constraints for combined open and closed world reasoning.

In order to prescribe how a mapping type must be used, SPIN is used as a constraint language. As described in Sect. 4.3.1, SPIN allows for storing SPARQL queries as RDF. However, additional properties, such as spin:constraint and spl:Attribute[15] enable the definition of prescriptions such as attributes constraining the usage of certain properties in the context of a specific class. In the following listing, we show how SPIN can be used to express the examples of constraints we already introduced above in natural language. It states that a rvl:PropertyMapping always maps exactly one rdf:Property to exactly one viso-graphic:GraphicRelation:

```
rvl:PropertyMapping
  spin:constraint
   [ a spl:Attribute ;
     rdfs:comment "There has to be exactly one target
                  graphic relation."^^xsd:string ;
     spl:maxCount 1 ;
     spl:minCount 1 ;
     spl:predicate rvl:targetGraphicRelation ;
     spl:valueType viso-graphic:GraphicRelation
   ] ;

  spin:constraint
   [ a spl:Attribute ;
     rdfs:comment "There has to be exactly one source property
                  of type rdf:Property."^^xsd:string ;
     spl:maxCount 1 ;
     spl:minCount 1 ;
     spl:predicate rvl:sourceProperty ;
     spl:valueType rdf:Property
   ] .
```

Attributes encapsulate a SPARQL query, which can be evaluated to decide whether some property is used as required. The most simple kind of query is a SPARQL ASK query – when it returns *yes*, the constraint is violated, when it returns *no*, the RVL model meets the constraint. The listing below shows such a constraint that is simply stored as a SPARQL ASK query. For better readability we show the SPARQL query in the usual syntax, not as SPIN, i. e., stored as RDF triples. The constraint ensures that whenever an rvl:ValueMapping defines a set of source values, it may not define a range of source values at the same time.

```
rvl:ValueMapping
  spin:constraint
   [ a sp:Ask ;
     rdfs:comment "You can either define a set of source values
                  or give bounds"^^xsd:string ;
     sp:where (

       # ... SPARQL-query represented as RDF statements with SPIN,
       # here shown as plain SPARQL ...

       ?this rvl:sourceValueSet ?svs .
       ?this rvl:sourceValueInterval ?svi .
      )
   ] .
```

More examples of constraints on RVL mappings, including such constraints that require a knowledge base conforming to the *VISO/facts* module can be found in Chapter 8, were we also provide a classification of the various kinds of constraints used in the OGVIC context as an overview.

[15] http://spinrdf.org/spl# is a SPIN library to provide common functions.

7.12 Conclusions and Future Work

This chapter introduced RVL, a novel declarative visualisation language for RDFS/OWL. Furthermore, we described the process of creating the language and discussed choices regarding a schema language for RVL.

With RVL, we answer the research questions *Q-1* and *Q-1.1*, asking for a way to define composable and shareable mappings. While the design of RVL was driven by concrete mapping situations as they occurred in our case studies, the various possibilities of composing mappings and the application to three very different scenarios during our case studies supports the assumption that RVL allows for flexible mapping solutions and is not bound to a specific domain. Yet, RVL has to be further evaluated in other domains possibly leading to extensions or changes.

Beyond the usage of RVL within the OGVIC visualisation approach for storing and composing visual mappings, we designed RVL to be generally useful as an exchange format for visual mappings. Further, RVL enables domain ontology authors to propose visualisation settings for RDFS/OWL data and store these settings side by side with the data.

RVL is used in the prototypes of visualisation systems that we present in detail in Chapter 8. Two of them are realised on top of an existing ontology editor, which is turned into an RVL specific editor by evaluating the RVL schema. The first one builds on TopBraid Composer and reads the RVL SPIN constraints to guide the user when defining RVL mappings. Furthermore, SPIN rules are used to process RVL mappings to the AVM model (cf. Chapter 6). The second one is realised as an OntoWiki Plug-In and directly processes RVL mappings to D3.js[16] using the SPIN API for constraint checking. The third one does not evaluate RVL constraints, but aims at implementing as much as possible of the RVL specification.

In the current version, RVL already covers all general requirements of Sect. 7.1, with two exceptions. First, although concepts for integrating simple interactions *(LR-3)* exist, these have not been fully specified and tested so far. Second, the extensibility of mappings *(LR-11)* is given to a certain degree, since, as for every RDF resource, third parties can add additional triples to mappings published on the web. However, like for CSS, there is no extension mechanism that would allow for defining (multiple) variants of mappings published on the web by specifying additional settings or removing unwanted settings.

Evaluating the current RVL against the sketches from our case studies (Sect. 3.3), we find that most of the sketches can already be described in RVL, including many of the more complex ones that can be composed from multiple mappings. In many cases, styles need to be selected in addition to visual mapping; Assigning a CSS style class to selected RDF resources can be done with Fresnel (Sect. 4.3.1) therefore, an integration of RVL with Fresnel and CSS should be considered. Besides this, we need to allow for describing static parts of graphics and mixing them with the ones that are generated by visual mapping. When generating texts, often, static strings are concatenated with dynamically generated strings, which is quite simple due to the linear, one-dimensional character of text. For graphics, having a graph-based structure, this is more complicated. This language feature is required when mappings introduce new graphic objects, for example, when using complex *Labelling* (e. g., the label composed from a clock and text in Fig. 3.4a).[17] Table 7.3 gives an overview of the visualisation cases that can be handled by the current version of RVL.

We encouraged researchers in the field of Semantic Web visualisation to comment on an alpha version of RVL [Pol13], add further use cases and point to inconsistencies in the specification. To ease the communication, we set up a detailed documentation of RVL on the web and allowed for discussing each language construct. The documentation can be accessed by pointing a web browser to the language's URI (http://purl.org/rvl/).

[16] D3.js (Data Driven Documents). http://d3js.org/, accessed: 02.07.2015.

[17] Currently, such actually static parts have to be created using simple mappings as a workaround.

Number	Description	RVL	Explanation
VC-1	Create a graphic object per resource	x	implicitly by PropertyMapping (Sect. 7.2.2)
VC-2	Map to Graphic Attributes	x	PropertyToGraphicAttributeMapping (Sect. 7.2.2)
VC-3	Map to Graphic-Object-to-Object-Relations	x	PropertyToGraphicObjToObjRelMapping (Sect. 7.2.2)
VC-4	Create additional graphic objects	(x)	implicitly: connectors, labels, . . .
VC-5	Define simple interactions	(x)	partially: e. g., co-highlighting (Sect. 7.9)
VC-6	Simplify the ontological model	x	simplifications (Sect. 7.2.7)
VC-7	Reuse / extend / compose mappings	(x)	reuse and composition (Sect. 7.10)
VC-8	Use complex standard graphics	–	
VC-9	Refer to parts of the graphic	x	by roles (e. g., viso-graphic:connector) (Sect. 7.10)
VC-10	Draw legends and labelled axes	(x)	legend flag on Mapping class; axes not yet supported
VC-11	Define styles	–	(could be done via future Fresnel integration)
VC-12	Benefit from good defaults	x	defaults specified for many mapping types (e. g., for calculating Value Mappings, Sect. 7.3)

Table 7.3: Overview of the visualisation cases that can be handled by the current version of RVL.

Chapter 8

The OGVIC Approach to Ontology-Driven, Guided Visualisation Supporting Explicit and Composable Mappings

The prerequisites for the approach have been laid in the previous chapters – we introduced a formalisation of graphic concepts by means of the Visualisation Ontology (VISO) and suggested the RDFS/OWL Visualisation Language (RVL) for defining visual mappings based on Semantic Web technologies. As a platform-independent model, we introduced the Abstract Visual Model (AVM). At this point we have all models available that we need to describe the overall visualisation approach.

The OGVIC approach can be classified as a model-driven [Tru06] and generative [CE00] approach, with multiple transformations between models and from models to code. However, its technological space is not the classical metamodelling technological space as described by the OMG, but it builds on RDF-based technologies to ease the consumption of RDF data and the production of settings (mappings) and models that can easily be exchanged in a Semantic Web context. With respect to the classification by configurability (Sect. 4.1.2), the OGVIC approach can be classified as »customisable via a UI«. Further, all aspects of customisation are covered – data, presentation and control.

A schematic overview of an architecture for the OGVIC approach has already been presented in the introduction (Fig. 1.4) to provide context for the contributions of this thesis. In the following we recall this architecture, add more details, and describe how the various models and processes that we introduced are connected. The architectural overview of OGVIC (Fig. 8.1) shows a system of models and transformation processes starting from the source data and finally resulting in a model (or code) for a concrete platform. Like the *Visualisation Reference Model* (Sect. 2.1.4), it is not an automatic, linear pipeline, but an incremental process, where intermediate results are taken into consideration. While not explicitly shown in this figure, all process steps, except e1 and e2, are meant to be modified interactively by users of the visualisation system.

Looking at the models used in our approach, not only the source data is represented as an RDF model, but there are four models (including the VISO ontology), each of which plays an important role at (visualisation) runtime. The first model holds the *source data* (c) from various domains. In a real world scenario, it may be selected from a Linked Data source or provided from a local file or triple store (a). A second model represents the current *RVL mappings* model (h) conforming to the RDFS/OWL Visualisation Language (RVL). The third model is the *VISO* ontology (g1–g3). For the OGVIC approach the modules data, graphic, facts and a fact base are required. In our example, we use the VISO module *VISO/facts/empiric* to provide knowledge from empiric visualisation studies, but this may be replaced by other fact bases, e. g., if new experiments lead to new insights. The AVM (f) is the fourth RDF model. In terms of the MDA, the AVM is a *Platform-Independent Model* (PIM). It consists of graphic objects and relations that have been formalised in the *VISO/graphic* module.

Looking at the processes, we can identify five main process steps: Selection (b), filtering (d), the guided editing of mappings (i), the generation of the AVM (e2) and, finally, the generation of code for a concrete platform based on the AVM (j). Step e1 resolves RVL convenience constructs into basic RVL constructs. As announced earlier, selecting (b) as well as filtering the source data (d) are indispensable parts of a productive visualisation system, but not discussed in this thesis as many solutions have been described for these process steps. Methods such as faceted browsing [ODD06, VWPM12, HZL08] or other (visual) querying and filtering approaches may be integrated to reduce the data before applying the remaining process steps. The *guided editing of visual mappings* (i) is detailed in Sect. 8.1. During this step not only the available data and the possible graphic relations are taken into consideration, but also the AVM needs to be available for introspection, since additional mappings may be constrained by existing ones. Unlike the guided mapping process, the actual transformation *generating* the AVM from the source data and the RVL mappings, is supposed to be automatic and unattended (e2). It can be seen as a PIM-to-PIM-transformation in terms of the MDA. Finally, in a further automatic transformation step – here called *rendering* (j), the code describing a graphic on a concrete platform is created from the AVM (PIM-to-PSM-transformation[1]). This may be a format such as SVG, HTML or X3D, which can then be directly displayed by a browser or – as it is the case for our third prototype – a format that can be further interpreted, e. g., by D3.js.

Extending the list of actors from Chapter 3, we introduce an *OGVIC developer* and a *platform developer* as well as a *visualisation user*. Not all components of this architecture have to be implemented by the same actor. The *OGVIC developer* may implement new RVL features, filtering mechanisms and guidance capabilities. The *platform developer* only has to know about the AVM and can start the development from this point. Her exclusive task is to develop a transformation to a specific platform such as D3.js or X3D. Finally, the *visualisation user* does not have to write program code or RVL mappings, but watches and interacts with visualisations configured by *visualisation authors* to explore or analyse data. A single user may play multiple roles at the same time.

Before discussing lessons learned from building three prototypes in Sect. 8.3–8.6, in the first two sections of this chapter, we deal with the core aspects of the OGVIC approach. We begin with the title's parts *ontology-driven* and *guided* in Sect. 8.1 and then explain the further aspects of *explicit* and *composable* mappings in Sect. 8.2.

[1] *Platform-Specific Model* (PSM).

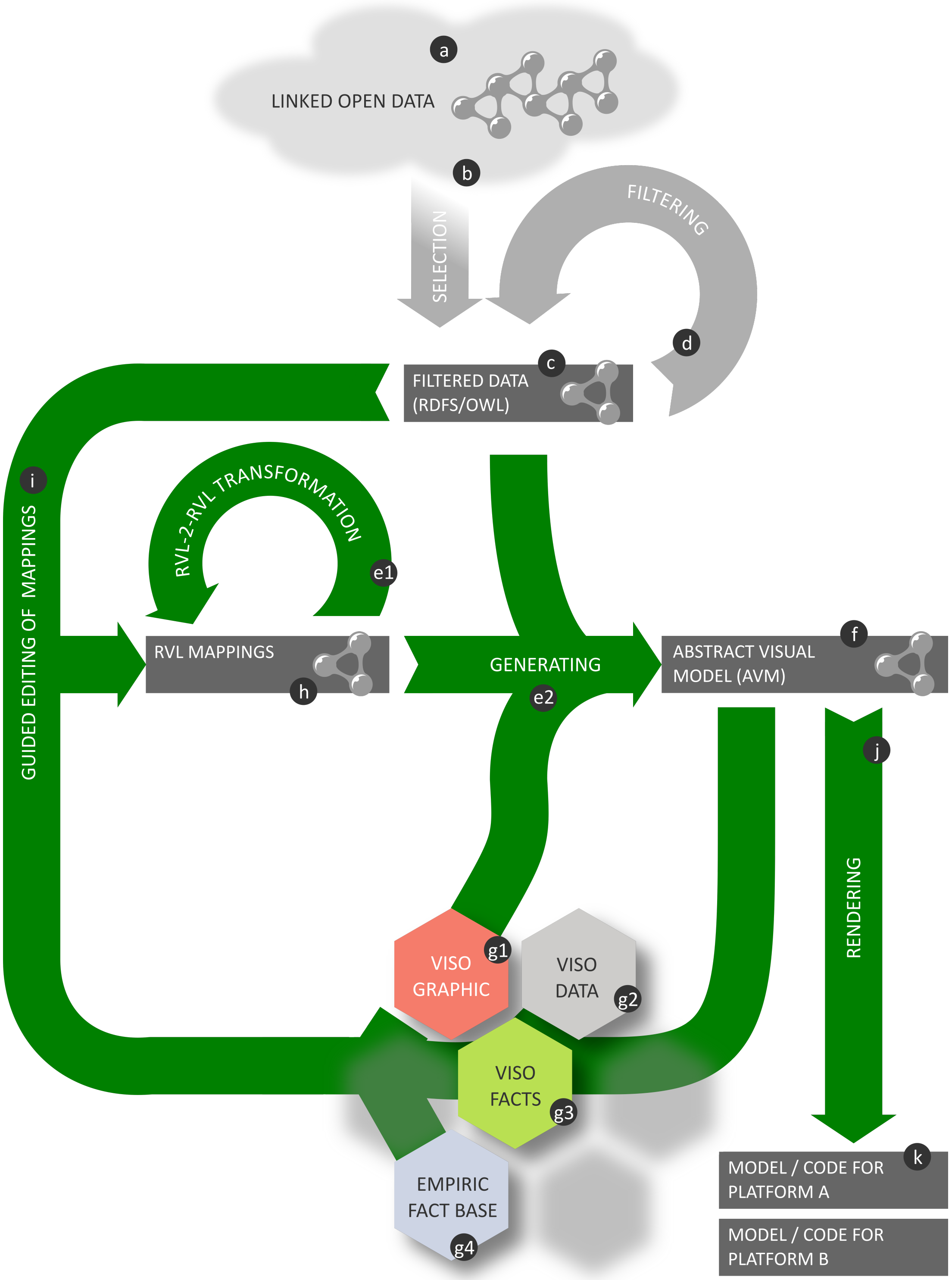


Figure 8.1: Suggested architecture for a visualisation design system implementing the OGVIC approach. The process steps and models labelled a, b, c, . . . g1, g2, . . . are referenced in the main text.

8.1 Ontology-Driven, Guided Editing of Visual Mappings

Offering the visualisation author *guidance* for the *visual mapping* process – i. e., for the editing of RVL mappings – is at the core of the OGVIC approach and corresponds to research question *Q-2*. With respect to the classification of guidance into flow guidance and step guidance (cf. Sect. 2.3) our approach focuses on step guidance for the *visual mapping* process step. Flow guidance plays a subordinated role in our approach; the technical processing order is given by the necessary pipeline (e. g., visual mapping first, then rendering) and user interaction can take place at any process step and in abitrary order, (e. g., filter first, then create visual mappings, then filter again). Guidance for the visual mapping process step (Fig. 8.1, i) covers the following:

- *mapping data relations to graphic relations* (including attributes such as colour or texture, but also graphic relations building complex spatial visual structures)
- *mapping data values to graphic attribute values*

To realise guidance under the OGVIC approach, we suggest to generate an ontology-driven, guided editor for visual mappings. Why should this UI be ontology-driven? In Sect. 7.11, we already anticipated that the RVL schema should be defined such that tooling could be derived from it and justified this by the reduction of redundancy and the increased extensibility of the RVL language. Since not only the RVL schema, but also facts and rules based on the VISO ontology need to be considered in the guidance process, we speak of *ontology-driven guidance*.

This also means that guidance needs to be provided based on constraints from different *sources* and serves different *purposes*. For example, some constraints ensure that mappings are valid, i. e., they conform to the RVL schema. Other constraints go beyond this to support guidance for questions of human perception or to discourage second-best mappings in favour of the most effective ones. Guidance can also employ different *levels* of enforcement from simple warning messages to carefully adapted error-prevention. In the following, we give an overview of the different constraint classes, as well as on the different levels of guidance that we distinguish. Finally, we show where and how ontologies are used for driving this process in the OGVIC approach and give examples of the defined constraints.

8.1.1 Classification of Constraints

A first idea was to classify constraints for guidance into syntactic and semantic constraints. However, since the distinction of semantic vs. syntactic constraints »*is somewhat arbitrary*« [SC02] and semantic constraints can be transformed into syntactic ones by refining the grammar, we instead classify the constraints by the following facets:

- Structural depth – The distinction into *intra-object* and *inter-object* can also be found in the context of database integrity constraints [Tü99]:
  - SINGLE PROPERTY – The constraint is on the value of some property, without referring to other properties
  - INTRA-OBJECT – The constraint references values of other properties associated with the same subject
  - INTER-OBJECT – The constraint references values of properties associated with other resources
- Inference needed
  - INFER – Standard (RDFS/OWL) reasoners can be used to infer subclasses and types of resources used in the constraints
- Usage of external knowledge

  - EXTERNAL TYPE SYSTEM – Usage of an existing type system for constraint definition
  - EXTERNAL FACTS – Usage of facts from an external knowledge base

- Severity
  - ERROR
  - WARNING
  - DISCOURAGED
  - RECOMMENDED

As suggested by the severity level »RECOMMENDED«, constraints can not only be used for validation purposes, but can also be used for different levels of guidance, which we discuss in the next section. Among the constraints based on external facts from VISO, we can further distinguish between expressiveness and effectiveness constraints, as well as between constraints concerning the usage of a single graphic relation and those concerning the composition of multiple graphic relations. This further visualisation-specific classification is not listed here; see Sect. 5.7 for details.

8.1.2 Levels of Guidance

We further classify guidance by the level of enforcement and the techniques used. This is strongly connected to the *Severity* facet of the constraint classification above.

- Display global messages, whenever a constraint is violated
  - ERROR MESSAGES – Display global error messages
  - WARNING MESSAGES – Display global warning messages
  - RECOMMENDATIONS – Display global recommendations
- Adapt the user interface
  - PREVENT VIOLATIONS – Constraint violations are not possible by construction of the editor
  - DISCOURAGE VALUES – Constraint violations when selecting values are possible, but marked and visually discouraged (by greying-out second-best options and pointing to constraint violations, e. g., with warning icons or marked red)
  - RECOMMEND VALUES – Best values are recommended, e. g., by highlighting or sorting, but no warnings appear
  - DISCOURAGE PROPERTIES – Constraint violations when selecting properties are possible, but marked and visually discouraged
  - RECOMMEND PROPERTIES – Best properties are recommended
  - DERIVE QUICKFIXES – Executable solutions are offered that solve the constraint violation

8.1.3 Implementing Constraint-Based Guidance with SPIN and VISO/facts

We already introduced simple SPIN constraints in Sect. 7.11.1 as a means to define a schema for RVL. In the absence of a standard to define contraints and hints for guidance, we also use SPIN to formulate the necessary (additional) constraints for guidance. In the following, we apply the classification of constraints and guidance levels to the OGVIC approach and give examples of constraints we implemented with SPIN and interpreted with our prototypes. SPIN constraints are attached to classes of the RVL language using the property spin:constraint. The constraint itself is a SPARQL query in SPIN notation (stored as triples). In most examples, to maintain readability, we give the query as plain text, though.

Structural Depth

With respect to the structural depth of the constraints, we require »SINGLE PROPERTY« (e. g., to constrain a datatype property to xsd:int) and »INTRA-OBJECT« constraints. An example for an »INTRA-OBJECT« constraint is given in Listing 8.1, where two properties of the same object are used in one constraint – rvl:lowerBound and rvl:upperBound – to express that rvl:lowerBound must always have a value smaller than rvl:upperBound.

```
rvl:Interval
    spin:constraint
        [ a sp:Ask ;
          rdfs:comment "Upper bound may not be less than lower bound." ;
          sp:where (

                # ... SPARQL-query represented as RDF statements with SPIN,
                # here shown as plain SPARQL ...

                ?this rvl:lowerBoundIncl ?lb .
                ?this rvl:upperBoundIncl ?ub .
                FILTER (?ub < ?lb) .
          )
        ] .
```

Listing 8.1: Intra-object constraint realised in SPIN. The constraint detects an error, if the upper bound of an rvl:Interval is less than the lower bound (corresponds to Fig. 8.6 in Sect. 8.3).

So far we could not identify cases where »INTER-OBJECT« references are required.

Inference and Constraint Evaluation

Constraints defined with SPIN can be checked with the open-source SPIN API (cf. Sect. 2.2.7 for background on SPIN-related software) and are internally evaluated using a SPARQL engine. Still, sometimes the preceding run of an RDFS reasoner is required. This may be necessary to extend constraints to subclasses and consider subproperty hierarchies. For example, using standard reasoning avoids the necessity to put a constraint for each subclass of rvl:PropertyMapping, since constraints defined for the class are automatically applied to subclasses. The benefits of combining (OWL and RDFS) reasoning with constraint checking – as it possible with SPIN – have been recently elaborated by Bosch et al. [BANE15].

Usage of External Knowledge

Referencing an »EXTERNAL TYPE SYSTEM« is the common case for the set of constraints that we defined for the OGVIC approach. For example, types of graphic relations defined in *VISO/graphic* are referenced in RVL constraints (cf. Listing 8.1). However, we can split the overall set of constraints into two subsets by the criterion »EXTERNAL FACTS«. We stored the two subsets separately as *RVL-Constraints* and *RVL-VISO-Constraints*.

The module *RVL-Constraints* provides all constraints that serve to describe valid RVL mappings, but does not need to evaluate facts from external knowledge bases. These constraints can been seen as part of a formal specification of the RVL language that we refer to as RVL schema in Sect. 7.11. Examples of such constraints are the constraint on rvl:ValueMapping given in Sect. 7.11 (last listing) and Listing 8.1.

The module *RVL-VISO-Constraints* provides all constraints that reference external facts from a knowledge base. These constraints go beyond ensuring valid RVL mappings but can be used for generating warnings and recommendations concerning the quality of the graphic. For example, we defined constraints on the expressiveness and effectiveness of visual mappings with respect to the scale of measurement of the data. An example of an expressiveness constraint is given in Listing 8.2. An example of an effectiveness constraint is given in Listing 8.3. The knowledge base utilised by *RVL-VISO-Constraints* has to conform to the *VISO/facts* module, i. e., it has to offer knowledge about graphics formulated with the *VISO/facts* vocabulary. As introduced in Sect. 5.7, we offer the module *VISO/facts/empiric* as a basic instance of such a knowledge base.

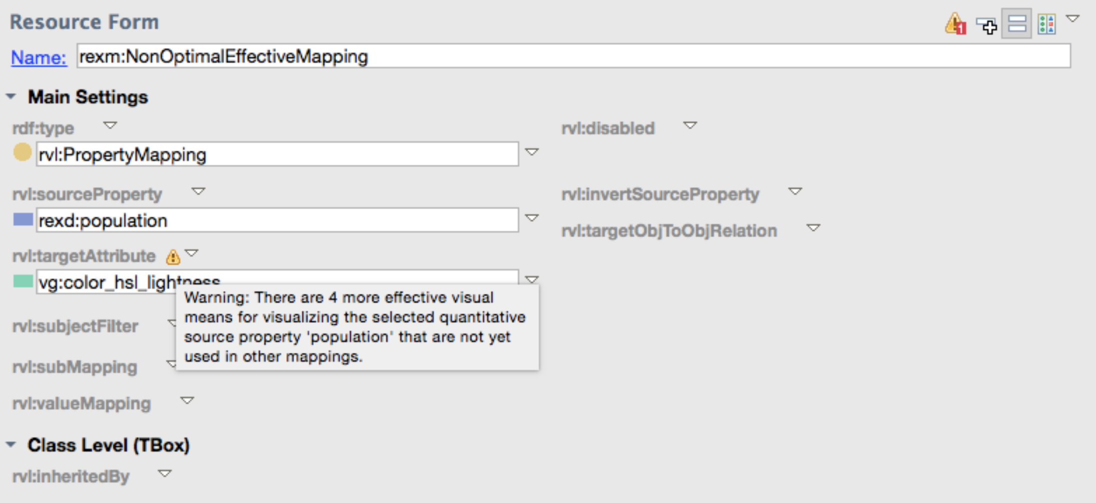
```
ASK WHERE {
    ?this rvl:sourceProperty ?sp .
    ?this rvl:targetGraphicRelation ?tgr .
    {
        ?sp rdfs:subPropertyOf viso-data:has_nominal_value .
        ?tgr viso-facts:not_expresses viso-data:Nominal_Data .
    } UNION {
        ?sp rdfs:subPropertyOf viso-data:has_ordinal_value .
        ?tgr viso-facts:not_expresses viso-data:Ordinal_Data .
    } UNION {
        ?sp rdfs:subPropertyOf viso-data:has_quantitative_value .
        ?tgr viso-facts:not_expresses viso-data:Quantitative_Data .
    }
}
```

Listing 8.2: Example of how an expressiveness constraint can be realised in SPIN (showing only the encapsulated SPARQL query as plain text).

```
CONSTRUCT {
    _:b0 a spin:ConstraintViolation .
    _:b0 spin:violationRoot ?this .
    _:b0 spin:violationPath rvl:targetAttribute .
    _:b0 rdfs:label ?cvLabel .
    _:b0 spin:violationLevel spin:Warning .
    _:b0 spin:fix _:b1 .
    _:b1 a :DeleteTriple .
    _:b1 rdf:subject ?this .
    _:b1 rdf:predicate rvl:targetAttribute .
    _:b1 rdf:object ?tga .
}
WHERE {
    ?this rvl:targetAttribute ?tga .
    {
        SELECT COUNT(?tga) AS ?altCount SAMPLE(?sp) AS ?sp1 SAMPLE(?som) AS ?som1
        WHERE {
            ?this rvl:sourceProperty ?sp .
            ?this rvl:targetAttribute ?tga .
            {
              ?sp rdfs:subPropertyOf viso-data:has_nominal_value .
              ?tga viso-facts:has_effectiveness_ranking_for_nominal_data ?rankValue .
              ?anyOtherGr viso-facts:has_effectiveness_ranking_for_nominal_data ?otherRankValue .
              BIND ("nominal" AS ?som)
            } UNION {
              ?sp rdfs:subPropertyOf viso-data:has_ordinal_value .
              ?tga viso-facts:has_effectiveness_ranking_for_ordinal_data ?rankValue .
              ?anyOtherGr viso-facts:has_effectiveness_ranking_for_ordinal_data ?otherRankValue .
              BIND ("ordinal" AS ?som)
            } UNION {
              ?sp rdfs:subPropertyOf viso-data:has_quantitative_value .
              ?tga viso-facts:has_effectiveness_ranking_for_quantitative_data ?rankValue .
              ?anyOtherGr viso-facts:has_effect.._ranking_for_quantitative_data ?otherRankValue .
              BIND ("quantitative" AS ?som)
            }
            FILTER (?tga != ?anyOtherGr)
            FILTER (?otherRankValue > ?rankValue)
```

```
      NOT EXISTS {
       ?anyOtherMapping rvl:targetGraphicRelation ?anyOtherGr .
       FILTER (?this != ?anyOtherMapping)
41    }
    }
    GROUP BY ?tga
  }
  FILTER (?altCount > 0)
46 BIND (fn:concat("There are ", ?altCount, " more effective visual means for visualizing
    the selected ", ?som1," source property '", afn:localname(?sp1), "' that are not yet
    used in other mappings.") AS ?cvLabel)
}
```

Listing 8.3: Example of how an effectiveness constraint can be realised in SPIN (showing only the encapsulated SPARQL query as plain text). The constraint definition includes a quickfix for deleting ineffective mappings using spin:fix (Line 7–11). Furthermore, the severity level – spin:violationLevel– is set to spin:Warning (Line 6). rvl-cs:DeleteTriple is a spin:UpdateTemplate for deleting arbitrary statements, which we instantiate here for deleting ineffective rvl:targetGraphicRelation-statements. Possible improvements include extending the quickfix to suggest the most effective visual means as well as considering the fact that visual means may be used more than once if we use submappings. See Fig. 8.2 and 8.3 for an example of how this constraint is interpreted in prototype P1.



Resource Form

Name: rexm:NonOptimalEffectiveMapping

Main Settings

rdf:type: rvl:PropertyMapping

rvl:disabled

rvl:sourceProperty: rexd:population

rvl:invertSourceProperty

rvl:targetObjToObjRelation

rvl:targetAttribute: vg:color_hsl_lightness

Warning: There are 4 more effective visual means for visualizing the selected quantitative source property 'population' that are not yet used in other mappings.

rvl:subjectFilter

rvl:subMapping

rvl:valueMapping

Class Level (TBox)

rvl:inheritedBy

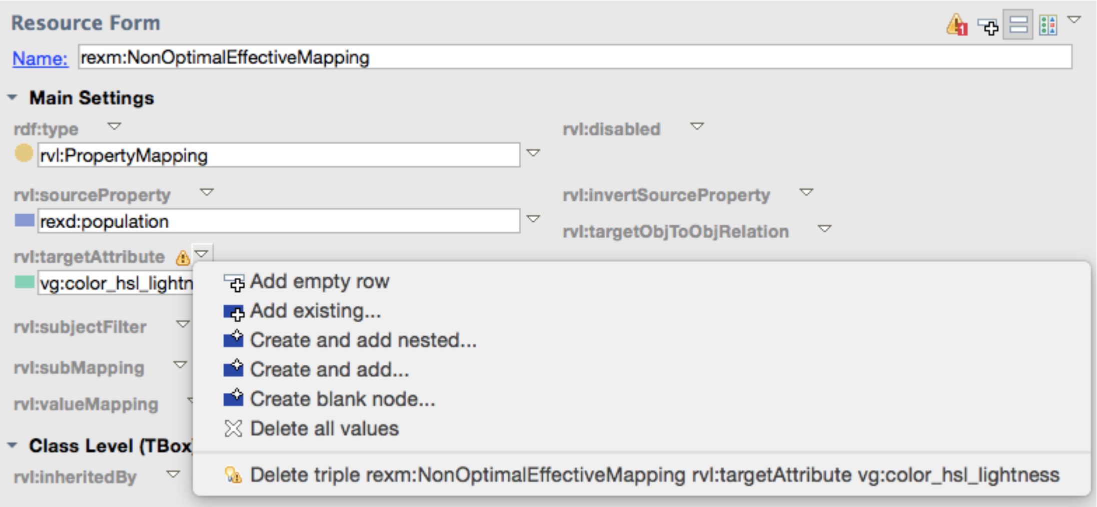
Figure 8.2: Effectiveness constraint – Warning: This figure shows how the constraint from Listing 8.3 is interpreted in the TopBraid-Composer-based prototype (see Sect. 8.3). A warning is issued for this property mapping, since viso-graphic:color_hsl_lightness was used for visualising the quantitative values of ex:population, which is not the optimal choice according to the effectiveness ranking that we formalised in the VISO module *VISO/facts/empiric*.

Supported Levels of Severity

Severity levels can be added to a SPIN constraint or SPIN rule by adding an extra triple using spin:violationLevel. For example, we assign the level spin:Warning if the effectiveness of a mapping is non-optimal and spin:Error if the expressiveness is not given.

Supported Levels of Guidance and Quickfixes

With respect to the levels of guidance used in the OGVIC approach, the levels »DISCOURAGE VALUES«, »RECOMMEND VALUES«, »DISCOURAGE PROPERTIES« and »DERIVE QUICKFIXES« have been employed. Our first (Sect. 8.3) and second protototype (Sect. 8.4) can display error and warning messages. These messages are not only listed in a global view, but also displayed locally on the involved property widgets of the respective constrained class (Fig. 8.2 and Listing 8.3). Furthermore, TopBraid Composer evaluates constraints to suggest possible values in the value-select-dialogues of the editor. However, errors are not completely avoided (»PREVENT VIOLATIONS«), nor are properties suggested »RECOMMEND PROPERTIES«. Listing 8.3 also gives an example of a *quickfix* defined using spin:fix. Fig. 8.3 shows how this quickfix is presented in TopBraid Composer.


Resource Form

Name: rexm:NonOptimalEffectiveMapping

Main Settings

rdf:type: rvl:PropertyMapping

rvl:disabled

rvl:sourceProperty: rexd:population

rvl:invertSourceProperty

rvl:targetObjToObjRelation

rvl:targetAttribute: vg:color_hsl_lightn

rvl:subjectFilter

rvl:subMapping

rvl:valueMapping

Class Level (TBox

rvl:inheritedBy

Figure 8.3: Effectiveness constraint – Quickfix: This figure shows how the constraint from Listing 8.3 is interpreted in the TopBraid-Composer-based prototype (see Sect. 8.3). A quickfix is offered to delete the statement that violates the effectiveness constraint.

8.2 Support of Explicit and Composable Visual Mappings

A second important feature of the OGVIC approach, are the aspects »explicit« and »composable« visual mappings, corresponding to research question *Q-1*. Composition is not an optional feature but required for realising most of the sketches from our case studies. Speaking of composition, we first have to clarify what is subject to composition. This includes the

- composition of graphic objects,
- composition of graphic relations,
- composition of mappings, and eventually, the
- composition of data as a pre-step (not in focus here).

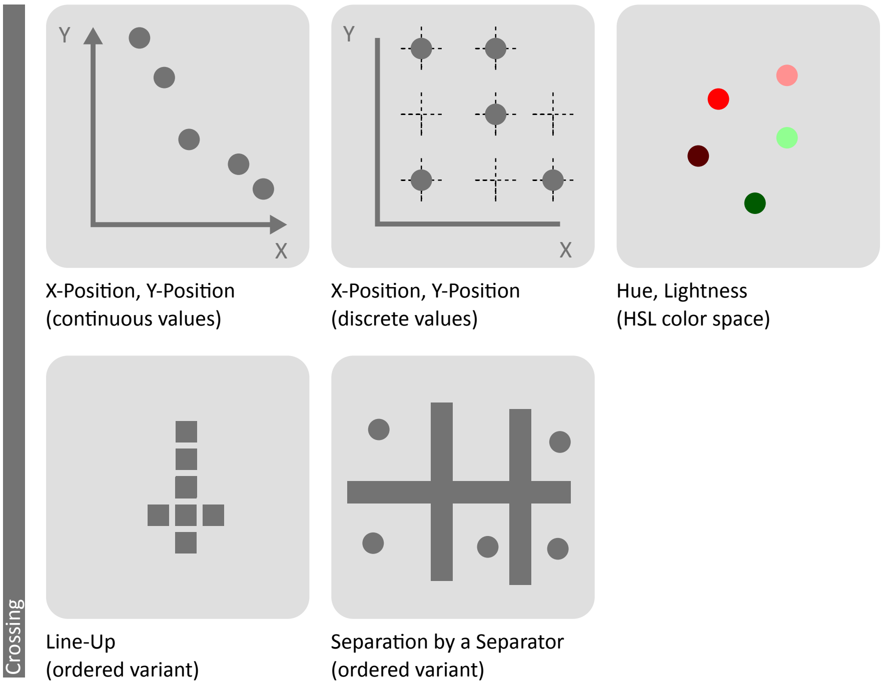
When introducing the AVM in Chapter 6, we already dealt with the composition of graphic objects (Sect. 6.5) and graphic relations (Sect. 6.6). We introduced how graphic objects as well as graphic relations and the emerging structures can be composed based on the syntactic graphic roles that graphic objects can play. Visual mappings may be composed as well. The prerequisite for reusing and composing mappings is that we make the visual mapping from data relations to graphic relations explicit. We provided the RVL language for this purpose, which defines the possible mappings. With the graphic module of the VISO ontology, we introduced a formal description for the graphic relations to be used within these mappings. Also the composition of visual mappings was already briefly mentioned in the context of the RVL language (Sect. 7.10): We showed how composability is realised by the submapping mechanism of RVL, in combination with the concept of the role-based AVM. In the following section, we discuss in more detail, which composition cases may occur, beyond the basic distinction we already described in the chapter on RVL – *Simultaneous Composition* and *Context Composition*.

8.2.1 Mapping Composition Cases

The following figures illustrate the identified cases of visual mapping compositions. We omit the concrete source data relations and focus on the composed graphic relations instead.

Simultaneous composition. All mappings are applied to the graphic objects independently of other mappings, i. e., removing one mapping does not influence the other mappings. Simultaneous composition steps are confluent, that means they can be performed in any order. Each mapping works on the same set of graphic objects. Fig. 8.4 shows an example of the simultaneous composition of multiple mappings. We use the term »simultaneous« referring to the »simultaneous combination of visual syntactic structures« described by Engelhardt [vE02].

Crossing of graphic attributes. A special case of a simultaneous composition is the crossing of graphic attributes. The same graphic attribute may be used multiple times if its value space can be split into multiple orthogonal dimensions. This works – and is frequently done – for *position* in physical space, but also for attributes like *colour*, which equally spans up a multidimensional colour value space[2].



X-Position, Y-Position
(continuous values)

X-Position, Y-Position
(discrete values)

Hue, Lightness
(HSL color space)

Line-Up
(ordered variant)

Separation by a Separator
(ordered variant)

Crossing of other graphic relations. Since graphic-object-to-object-relations consume graphic attributes like spatial position, crossing can also be applied to relations such as *separation by a separator* or *line-up*, as long as each mapping is constrained to use only one of the dimensions available. The last subfigure above shows an example of this: Two different

[2] For spatial dimensions, this works only if the graphic attribute comprises only one dimension, since already the crossing of two by two dimensions would result in four spatial dimensions, which is more than the three spatial dimensions that can be perceived by the human eye.

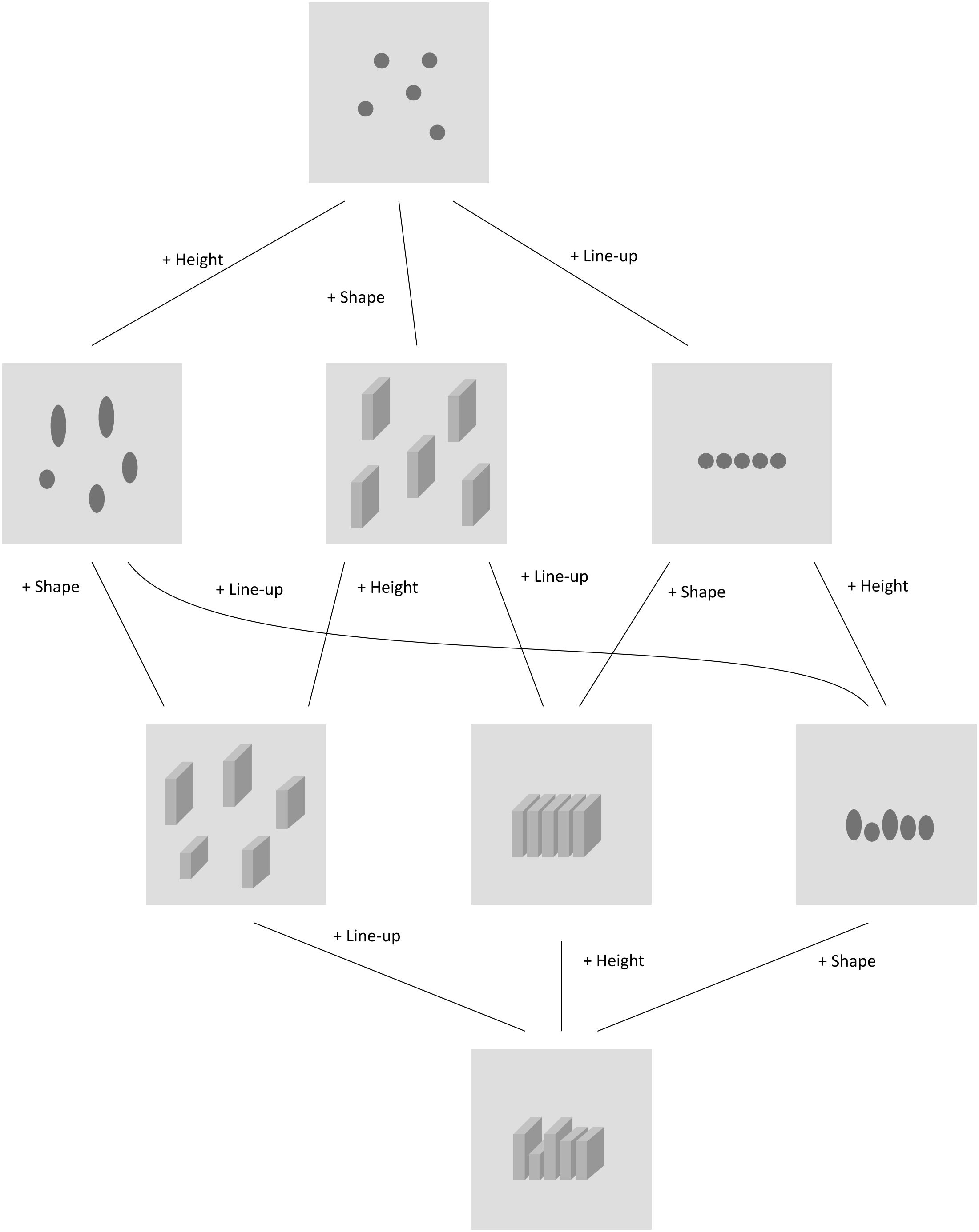
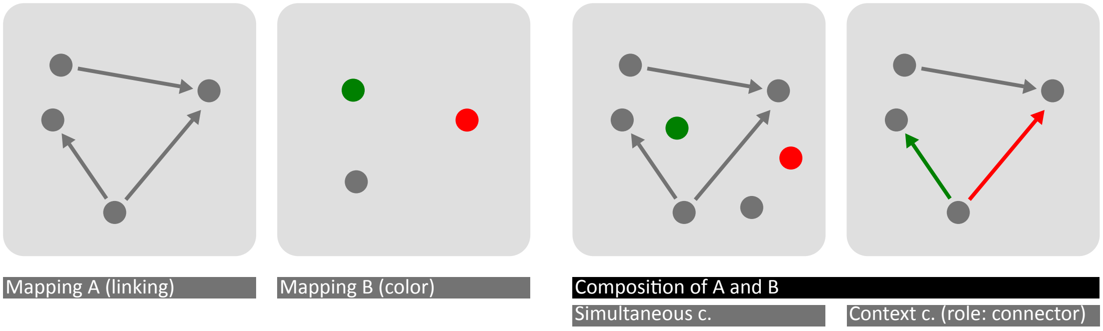


Figure 8.4: Example for a set of visual mappings that are *simultaneously* applied to the whole set of »raw« graphic objects. The order in which they are applied to the graphic objects does not affect the final result, since simultaneous composition is confluent. Because *line-up* is the only graphic relation that internally uses *position*, no conflicts occur. Depending on the data, this may be different if we compose with *containment* (cf. Sect. 5.7.3).

semantic relations are both mapped to the same graphic relation *separation by a separator*. This works, because one is mapped to horizontal separators and the other to vertical separators.

Context composition. In a context composition, the second mapping applies only to a subset of the graphic objects defined by the first mapping, the *context*. In the next subsection, we discuss multiple options to define a context. Two mappings taking part in a context composition cannot be changed independently of one another, because one mapping needs to work on graphic objects that depend on the other mapping. Sometimes, the objects to which a context composition is applied, may even have been newly created by the other mapping. In the example below, the second mapping is used in the context of the first mapping, which is determined by a graphic role *connector*. This graphic role is assigned during the first mapping, so the scope of the second mapping is limited by the first one. Still, the second mapping is self-contained and could be reused in other contexts.



8.2.2 Selecting a Context

Describing a context composition requires the selection of a graphic context as well as a data context:

Selecting a graphic context. In order to create a graphic context, we need to reference parts of a graphic. In RVL, the rvl:submapping_onRole property of submappings allows for selecting only those graphic objects that currently play a specific graphic role.

While not yet being part of the RVL specification, another option is to select the graphic objects indirectly via the resources they represent. Filters, which may even select a single entity by ID, could be applied to further restrict the set of graphic objects. Further contexts based on the *structure* of the data or the structure of the graphic are conceivable. Context could, for instance, be given through structural references by

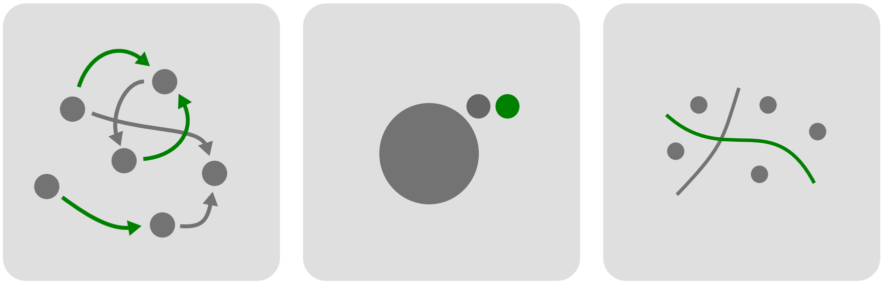
- a certain level of a hierarchy (DAG required),
- each level of a hierarchy (DAG required),
- leaf nodes of a hierarchy per branch (DAG required),
- second degree neighbours (this would be dependent on the »position« of a user, e. g., in a node-link diagram),
- nodes with fan-out $>$ 20.

Finally, further contexts, such as referencing specific graphic objects by their graphic values, are possible. For example, we could apply some mapping to each graphic object that was coloured »red« by a previous mapping. A similar case is to pick graphic objects by their ID. Both cases should be rare.

Selecting a data context. To determine the data to be processed by the composed mapping, we also need a data context. In RVL, we can refer to the statement processed by the »supermapping«. With the rvl:submapping_onTriplePart property, we then define whether subject, predicate or object of this statement should form the new data context for the submapping (cf. Sect. 7.10).

8.2.3 Using the Same Graphic Relation Multiple Times

Valuable graphic relations such as *linking*, but also *separation by a separator* or *proportional repetition* can be mapped more than once, if their instances can be clearly distinguished. This can be easily achieved if the respective graphic relation involves creating additional graphic objects (such as a *separator*, a *connector*, a *label* or *proportionally repeated objects*). In this case, a second mapping (e. g., to colour) can be applied on the newly created objects (by means of a context composition) to resolve the ambiguity. In the figure below we give three examples of using the same graphic relation for two different semantic relations: for linking, we colour the connectors; for proportional repetition, we colour the repeated objects; for separation by a separator, we colour the separators (here we use grey and green). A similar example from our case study sketches is RO-6 (Fig. 3.4a), where both ro:refines and ro:isInConflictWith are mapped to *linking*. Here, the two completely unrelated semantic relations are distinguished by the shape of the connectors. A special case of this principle is the use of colour/shape to distinguish subproperties in example CIT-1 (Fig. 3.4c) and CIT-5 (Fig. 3.3c).



Linking

Proportional Repetition

Separation by a Separator

With respect to the scalability and complexity of the composition approach, we have to keep in mind that while calculations on the composability of various graphic relations may quickly become complex and expensive, a single graphic will, for the sake of human perception, only make use of a limited number of relations. If it is necessary to encode more relations, multiple views will probably be used, which should also break down the necessary calculations into reasonable fractions.

Having discussed the defining features of the OGVIC approach, in the following, we present three prototypes that have been developed to implement aspects of our approach and discuss their differences and shortages. For each prototype (P1–P3), we give an architectural overview. These figures refer back to the schematic overview of an architecture for the OGVIC approach that we presented in the introduction (Fig. 8.1).

8.3 Prototype P1 *(TopBraid-Composer-based)*

A first prototype (P1) of the OGVIC architecture (Fig. 8.5) was realised based on the graphical ontology modelling suite TopBraid Composer. The modelling environment comprises a graphical editor for the RVL language and a SPIN-rule-based interpreter that takes RVL definitions and data and transforms them into the AVM. Since SPIN-rules are grounded in SPARQL 1.1 Update, they may be regarded as graph transformation rules. The tooling provided by the modelling environment based on the RVL schema (defined in RDFS+SPIN), can validate RVL mappings and issue warnings on the UI when the user violates a constraint. Since the constraints and rules can access external graphs such as the knowledge base *VISO/facts/empiric*, also the expressiveness and effectiveness of visual mappings can be checked. To allow for both standard inference (open world reasoning) and constraint checking with SPIN (closed world reasoning) in the same environment, chains of different reasoners can be specified. Beyond displaying error messages and warnings (Fig. 8.6), SPIN allows for defining quickfixes. Quickfixes (spin:fix) are realised as SPARQL Update requests, which can be evaluated to suggest changes that eliminate possibly detected constraint violations. Finally, the AVM is rendered to HTML based on UISPIN-templates (Sect. 4.3.1) within the boundaries of the limited visual structures that UISPIN supports.

As an example for a graph transformation with SPIN, we demonstrate how the convenience construct rvl:ResourceMapping (cf. Sect. 7.2.6) is internally transformed to an rvl:PropertyMapping and a connected rvl:ValueMapping. Listing 8.4 shows the original rvl:ResourceMapping, followed by the necessary transformation of the RDF graph described using spin:rule (Listing 8.5). Finally, the transformation result is shown in Listing 8.6. This exemplary transformation is part of the *RVL-2-RVL* SPIN-rules (step e1 in Fig. 8.5). Similarly, transformations from the RVL mapping graph to the graph representing the AVM (step e2) have been realised (cf. Listing 8.8 in Sect. 8.6.3).

It would have been desirable to build on top of P1 to benefit from the SPIN-based environment from TopBraid instead of building completely different prototypes. Unfortunately, this was not possible due to license restrictions. Since reimplementing a comparable generic open-source infrastructure was out of scope of this thesis, two other specific prototypes have been developed to implement further aspects of the OGVIC approach.

```
ex-mapping:ExampleResourceMapping
     a rvl:ResourceMapping ;
     rvl:targetGraphicRelation ex:exampleGraphicRelation ;
     rvl:sourceValue ex:exampleSourceValue ;
     rvl:targetValue ex:exampleTargetValue .
```

Listing 8.4: Resource mapping.

```
INSERT {
   GRAPH ?genRvlGraph {
      ?genPm a rvl:PropertyMapping .
      ?genPm rvl:sourceProperty rdf:ID .
      ?genPm rvl:targetGraphicRelation ?tgr .
      ?genPm rvl:valueMapping ?genVm .
      ?genVm a rvl:ValueMapping .
      ?genVm rvl:sourceValue ?sv .
      ?genVm rvl:targetValue ?tv .
   }
} WHERE { # continued on page 184 ...
```

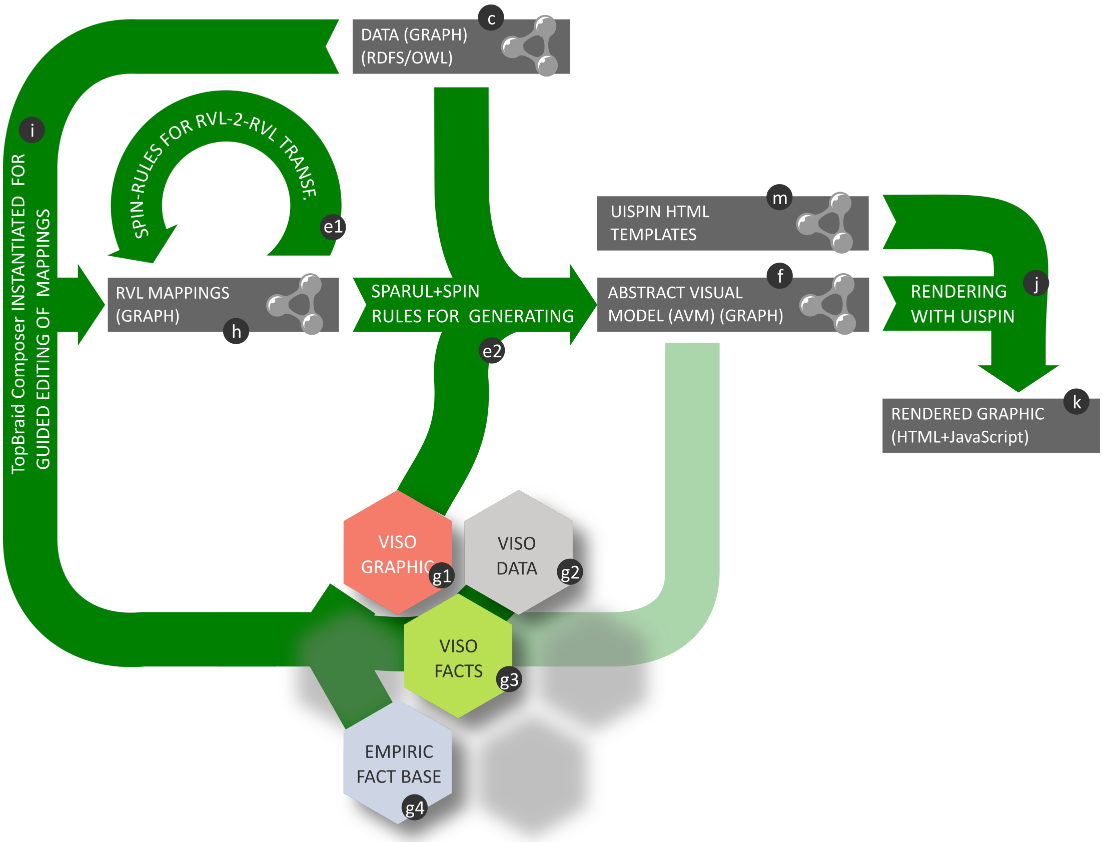
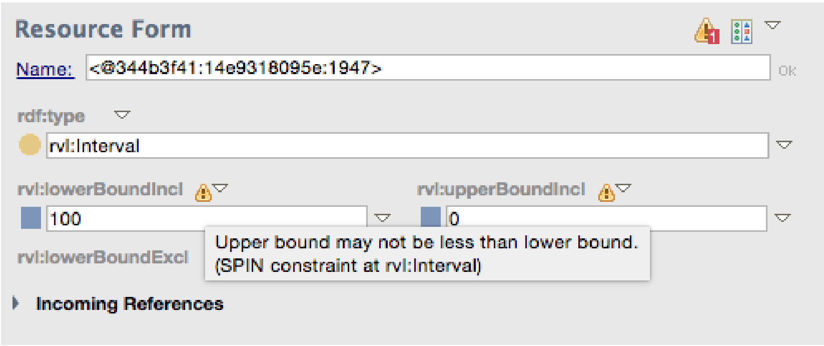


Figure 8.5: Architecture of the TopBraid-Composer-based prototype: RVL mappings can be (graphically) edited within the TopBraid Composer environment (i), offering basic guidance functionality. The TopBraid Composer SPIN engine is then used for interpreting RVL with SPIN-rules and constraints (SPIN-based graph transformation; step e2) and building the AVM (f) from c, h and g1–g4. Finally, the AVM is rendered (j) to HTML (k) based on UISPIN templates (m). In step e1, SPIN-rules are used in case it is necessary to transform from RVL to RVL to resolve RVL convenience constructs into basic mapping types.



Resource Form

Name: <@344b3f41:14e93180095e:1947>

rdf:type

rvl:Interval

rvl:lowerBoundIncl

100

rvl:upperBoundIncl

0

rvl:lowerBoundExcl

Upper bound may not be less than lower bound.
(SPIN constraint at rvl:Interval)

Incoming References

Figure 8.6: TopBraid-Composer-based prototype – Bounds constraint: Warnings are issued for this mapping, since the values for the lower and upper bound violate an RVL constraint stating that the upper bound value must be greater than the lower bound value (corresponds to Listing 8.1).

```
# .. continued from page 182

   BIND (<http://example.org/rvl/graph/> AS ?mappingsGraph)
   BIND (<http://example.org/rvl/generated/graph/> AS ?genRvlGraph)
   BIND (afn:localname(?tgr) AS ?localNameTgr)
   BIND (afn:localname(?rm) AS ?localNameRm)
   GRAPH ?mappingsGraph {
      ?rm a rvl:ResourceMapping .
      ?rm rvl:sourceValue ?sv .
      ?rm rvl:targetValue ?tv .
      ?rm rvl:targetGraphicRelation ?tgr .
   }
   GRAPH ?genRvlGraph {
         BIND (smf:buildURI(":GenPropertyMappingForID2{?localNameTgr}") AS ?genPm)
         BIND (smf:buildURI(":GenValueMappingForResourceMapping{?localNameRm}") AS ?genVm)
   }
}
```

Listing 8.5: Transformation from rvl:ResourceMapping to rvl:PropertyMapping and rvl:ValueMapping in SPIN (showing only the encapsulated SPARQL query of the spin:rule as plain text). We use the SPIN function smf:buildURI to concatenate a name for the generated mappings.

```
rvl-gen:GenValueMappingForResourceMapping
     a rvl:ValueMapping ;
     rvl:sourceValue ex:exampleSourceValue ;
     rvl:targetValue ex:exampleTargetValue .

rvl-gen:GenPropertyMappingForID2exampleGraphicRelation
     a rvl:PropertyMapping ;
     rvl:sourceProperty rdf:ID ;
     rvl:targetGraphicRelation ex:exampleGraphicRelation ;
     rvl:valueMapping rvl-gen:GenValueMappingForResourceMapping .
```

Listing 8.6: Generated property mapping and value mapping.

8.4 Prototype P2 *(OntoWiki-based)*

A second prototype (P2) is based on OntoWiki [DAR06], a generic ontology editor and infrastructure (Fig. 8.7). It consists of two plugins for OntoWiki and a constraint checker.

The first plugin adds a *Visualise* view to the OntoWiki tabs (Fig. 8.8). The view allows for filtering resources by type and for (de)activating single mappings. While mappings to graphic attributes can be freely combined, mappings to other graphic relations like *containment* can not, since the concept of RVL submappings is not yet supported in this prototype. Instead, the mapping to complete graphic types like *Collapsible tree* is offered.

The second plugin adds a basic guidance mechanism to the generic editor of OntoWiki. Whenever constraints defined in the RVL schema – including visual effectiveness and expressiveness constraints – are violated, a warning or error message gets displayed in the editor (Fig. 8.9).

The constraint checker wraps the SPIN API, which can evaluate the SPIN-constraints defined in the RVL schema. For the communication between OntoWiki (written in PHP) and the constraint checker (Java), a REST interface is used.

P2 was built by Pooran Patel as part of his master thesis [Pat13] and works with a preliminary version of RVL and VISO.

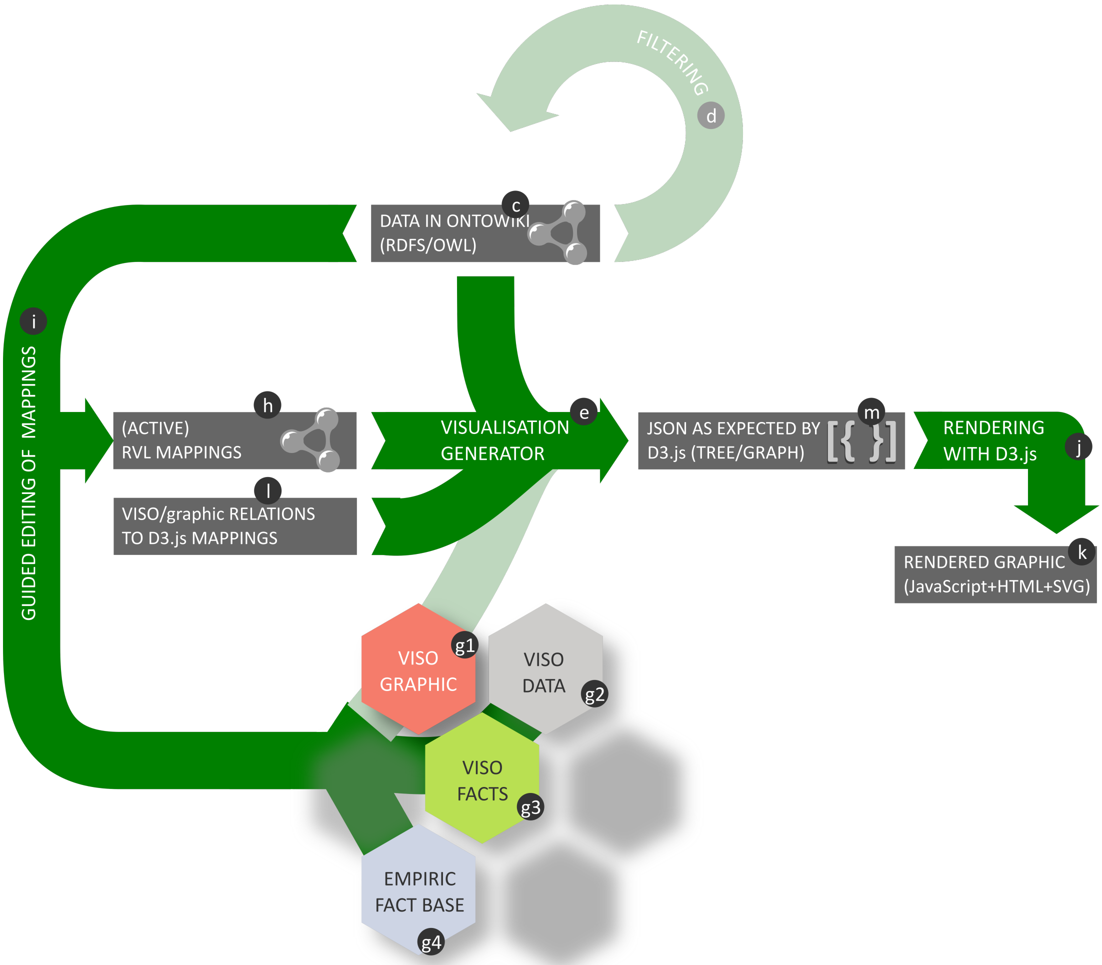
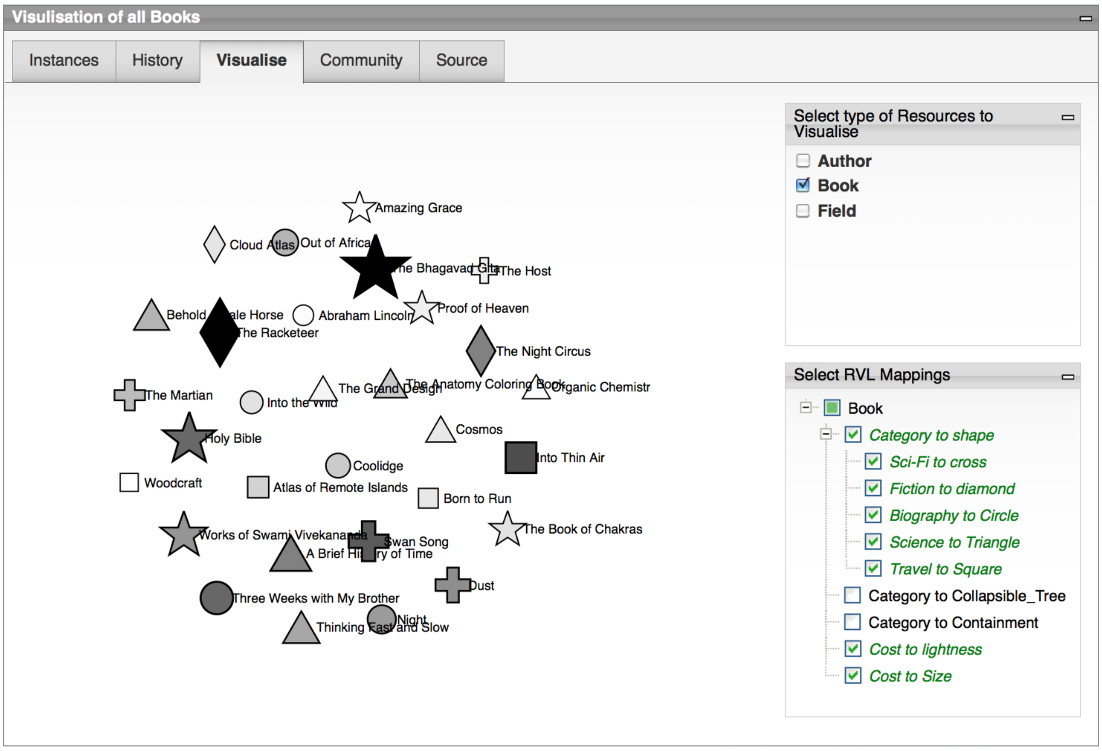


Figure 8.7: Architecture of the OntoWiki-based prototype: OntoWiki provides on-board data import (c) and generic filtering (d). RVL mappings (h) can be edited with the OntoWiki editor, which was extended to support basic VISO-based guidance functionality (g1–g4, i). Once the mappings are created, a *Visualisation Generator* (e) processes mappings, data and an additional mapping from VISO graphic relations to D3.js graphics (l) and prepares them for rendering with D3.js (m, j).



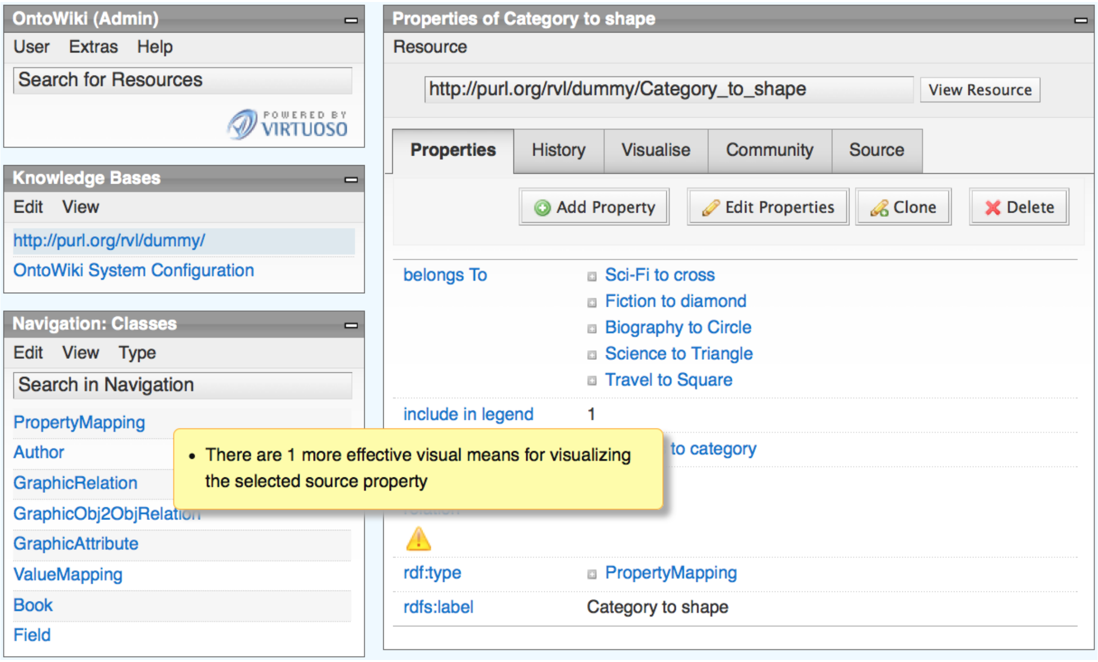
Select type of Resources to Visualise

☐ Author
☑ Book
☐ Field

Select RVL Mappings

- Book
  - ☑ *Category to shape*
    - ☑ *Sci-Fi to cross*
    - ☑ *Fiction to diamond*
    - ☑ *Biography to Circle*
    - ☑ *Science to Triangle*
    - ☑ *Travel to Square*
  - ☐ Category to Collapsible_Tree
  - ☐ Category to Containment
  - ☑ *Cost to lightness*
  - ☑ *Cost to Size*

Figure 8.8: OntoWiki-based prototype – Visualise view: A simple filtering of resources by type is realised (top right) and a list of Property Mappings and Value Mappings is shown (down right). Each mapping can be (de)activated separately. The resulting graphic (left) combines mappings to *lightness*, *size* (area) and *shape*. However, in P2 mappings to *containment* or *Collapsible Tree* cannot be combined with the other mappings. Also a mixture of mappings to graphic types (Collapsible Tree) and mappings to graphic relations (containment) is used.



http://purl.org/rvl/dummy/

OntoWiki System Configuration

PropertyMapping
Author
GraphicRelation
GraphicObj2ObjRelation
GraphicAttribute
ValueMapping
Book
Field

Resource: http://purl.org/rvl/dummy/Category_to_shape

belongs To	Sci-Fi to cross Fiction to diamond Biography to Circle Science to Triangle Travel to Square
include in legend	1
	to category
⚠	
rdf:type	PropertyMapping
rdfs:label	Category to shape

- There are 1 more effective visual means for visualizing the selected source property

Figure 8.9: OntoWiki-based prototype – Effectiveness warning: Warning icons are placed next to properties that have been set to non-optimal values with respect to *effectiveness*.

8.5 Prototype P3 *(Java Implementation of RVL)*

A third prototype (P3) does not focus on the guidance aspect but on interpreting as much of the RVL specification as possible and on rendering graphics from the AVM (Fig. 8.11). It combines an initial implementation of the RVL specification in Java with a generator for D3-compatible JSON[3] and a set of plugins (JavaScript) to bind the AVM »data« to D3 graphics. To ease experimenting with the prototype, a basic web front end was built (Fig. 8.12), which allows for rendering the use-cases defined in the analysis chapter. Furthermore, the user can enter RDF data, as it may have been downloaded from a SPARQL access point, and a set of RVL mappings. After interpreting the mappings, building the AVM and generating JSON for the D3 graphics, the rendered graphic is shown or updated.

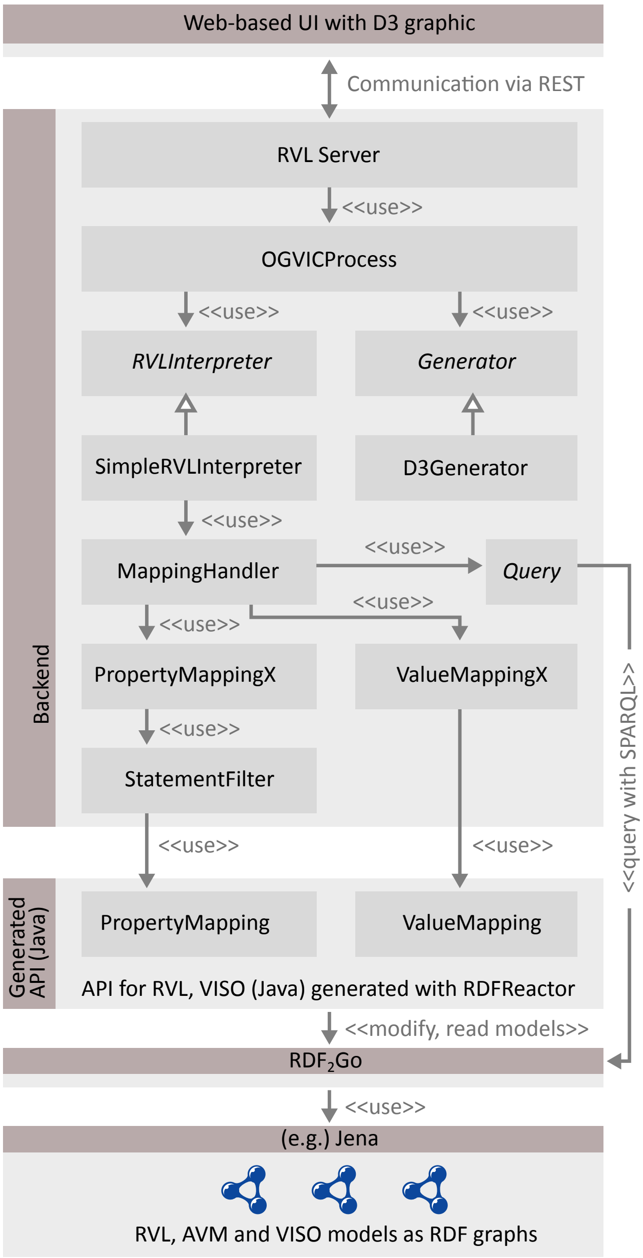


Figure 8.10: Prototype P3 – Detailed architecture and selection of important classes.

Following the more detailed architecture shown in Fig. 8.10 from top to bottom, we introduce the main parts of this implementation: The web-based UI can be used to process projects from a visualisation library or create and execute ad hoc projects. It communicates with the RVL Server, which offers the processing functionality via a REST[4] interface. The transformation pipeline is represented by an OGVICProcess. It can be fed with data and mapping files to provide a model of the data as well as a model of the RVL mappings. OGVICProcess orchestrates the classes RVLInterpreter and Generator. When running the process, the mappings are applied to the data and finally the generated platform-specific code can be retrieved from the process.

The RVLInterpreter manages the transformation from RVL mappings and data to the AVM and triggers the interpretation for all mappings in the mapping model, separated by the type of mapping. The actual transformation of each mapping is passed to MappingHandlers. Currently, the only implementation of

[3] JavaScript Object Notation (JSON). http://json.org/, accessed: 06.07.2015.
[4] Representational State Transfer (REST) is an architectural style for distributed systems.

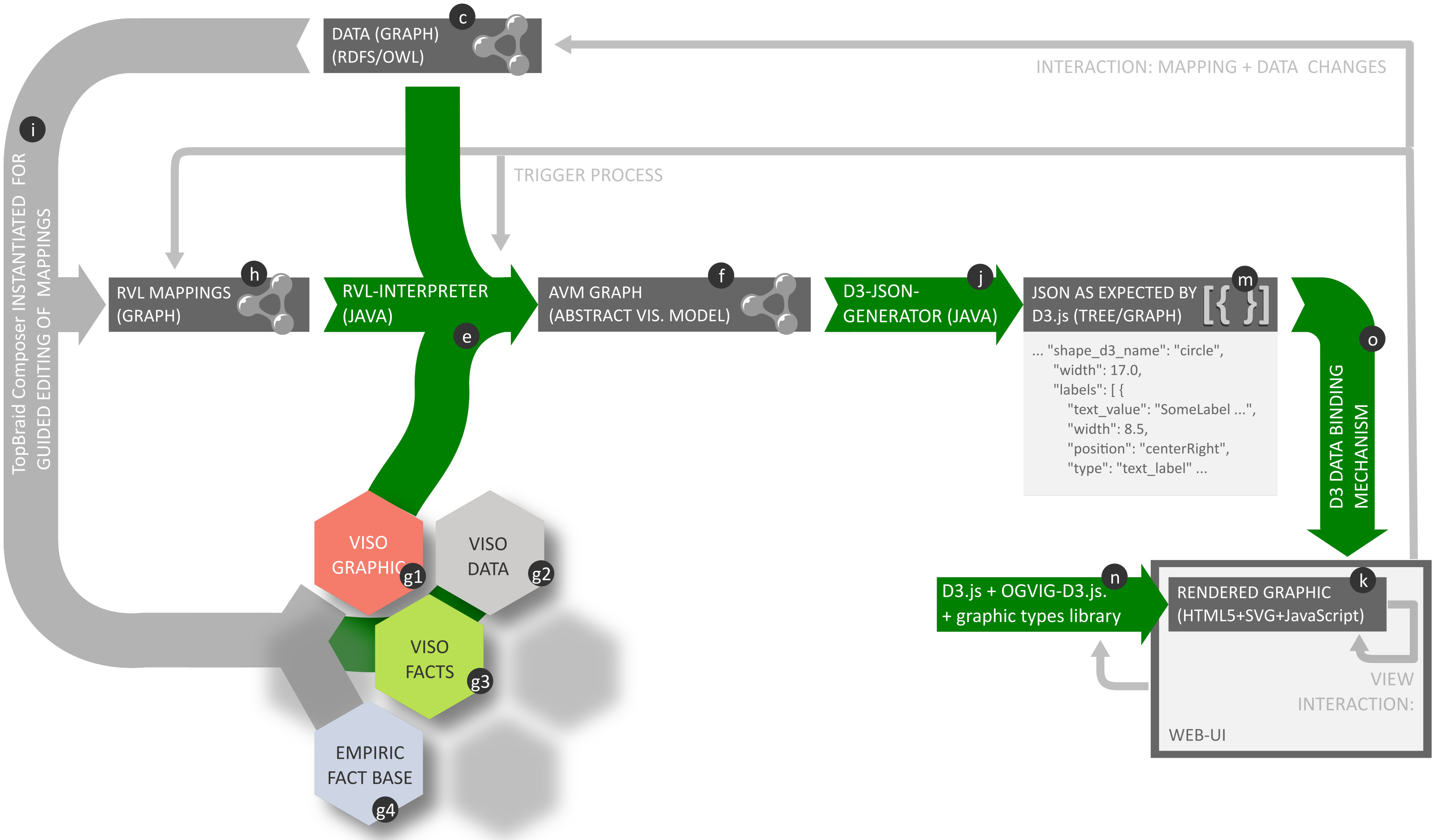
the RVLInterpreter interface is SimpleRVLInterpreter, which uses plain Java. Alternative implementations could use graph transformations described in SPIN or a similar language and work directly on the graph (see Sect. 8.6.3). For each graphic relation, a MappingHandler exists that performs the actual building of the AVM. Much of the functionality of the transformation can be shared between superclasses. In case a submapping is found, handlers can call other handlers to work on the submapping. Submappings will immediately be processed. The Query class eases the generation of SPARQL queries that are frequently needed to select mappings or data objects. Queries can be extended and configured to reduce redundancy. StatementFilter evaluates RVL filters, such as rvl:subjectFilter. Filters are used to constrain the application of a mapping to a limited set of subjects or objects as described in Sect. 7.2.

The class D3Generator implements a generator for D3.js-conform JSON data. It is meant to be only one of several generators. For example, a further subclass of Generator could be created to support X3D. We give an example of the JSON format in Listing. 8.7. Currently, two different variants of the JSON format need to be generated and a graphic type needs to be defined (cf. the discussion in Sect. 8.6.6).

We access the RDF graphs (representing the VISO, AVM and RVL models) in a domain-specific way using generated Java classes. Instead of adding statements to a graph, get-, add-,

```
{
  "nodes": [
    {
      "uri": "http://purl.org/rvl/example-data/Some_URI_of_some_resource",
      "roles":[
        "linkingDirected_startNode", "linking_node",
        "labeling_base"
      ],
      "shape": "circle",
      "width": 17.0,
      "labels": [
        {
          "text_value": "Some URI of some resource",
          "width": 8.5,
          "position": "centerRight",
          "type": "text_label"
        }
      ]
    }, ...
  ],
  "links": [
    {
      "shape": "arrow",
      "labels": [
        {
          "text_value": "partOf",
          "width": 8.5,
          "position": "centerRight",
          "type": "text_label"
        }
      ],
      "uri": "http://purl.org/rvl/example-data/partOf",
      "type": "LinkingDirected",
      "source_uri": "http://purl.org/rvl/example-data/Some_URI_of_some_resource",
      "target_uri": "http://purl.org/rvl/example-data/Another_URI_of_another_resource"
    }
  ],
  "graphic_type": "force-directed-graph"
}
```

Listing 8.7: Example of the JSON format used for processing the AVM with the OGVIC D3 plugins (non-hierarchical variant as expected, for instance, by the graphic type *Force-Directed Graph*).



```
... "shape_d3_name": "circle",
    "width": 17.0,
    "labels": [ {
        "text_value": "SomeLabel ...",
        "width": 8.5,
        "position": "centerRight",
        "type": "text_label" ...
```

Figure 8.11: Architecture of the P3 *(Java)* prototype: Since the focus of this prototype is on implementing the RVL specification, it offers no built-in data-import or filtering functionality. Also, we do not offer a guided (graphical) editor, but P1 may be used to edit an external RVL mapping file, which can then be imported (i). The *RVL-Interpreter* (e) builds an AVM (Abstract Visual Model; f) according to the declarative RVL mapping definitions (h), and consisting of terms from *VISO/graphic* (g1). The AVM is processed by the *D3-JSON-Generator* (j) to produce a D3-specific model in JSON notation (m). The D3-based OGVIC-d3.js library (n) is then used to render this model to an interactive HTML+SVG+JavaScript representation (k). Due to the D3 data binding mechanism (o), changes in the JSON model can trigger incremental, animated updates of the graphic. A very basic text editor allows for changing the underlying data and mappings (this requires regenerating the AVM and JSON model). Like in the *Visualisation Reference Model* (Sect. 2.1.4), we document, which steps can be interactively modified.

Process step	P1 *(TopBraid-Composer-based)*	P2 *(OntoWiki-based)*	P3 *(Java)*
Filtering data	–	(–)	–
Guided RVL editing	x	x	–
Generating RVL $\mapsto$ AVM	partially, via graph transformations with SPIN+SPARQL Update	–	x
Rendering AVM $\mapsto$ HTML+SVG	very basic, via UI-SPIN	RVL directly evaluated	x
View interactions	(x)	x	x

Table 8.1: Coverage of the three prototypes P1, P2, P3 with respect to the process steps described by the OGVIC approach.

and set-methods of Java classes representing the RDFS and OWL classes can be called. Instances of these classes are stateless, i. e., changes are passed back and exclusively stored in the RDF graph. For our specific case, this means, changes are stored exclusively in the AVM model. We use a code generator based on *RDFReactor* [Vö06] for creating plain Java classes as wrappers for the RVL (RDFS/OWL) classes. RDFReactor uses *RDF2Go* [Vö05] as an abstraction layer over concrete RDF frameworks like Sesame [SES] or Jena[5]. In cases where a SPARQL query is prefered to the generated API, the underlying graphs can still be directly accessed via RDF2Go.

8.6 Discussion: Lessons Learned from the Prototypes and Future Work

Table 8.1 gives an overview of the coverage of the three prototypes with respect to the process steps described by the overall OGVIC approach (also cf. Fig. 1.4 in the introduction). In summary, we can say that none of the prototypes covers the whole functionality of the OGVIC approach. However, many parts are covered by one of the three prototypes. For many aspects, a first step in order to improve the presented prototypes would be to combine the features from P1, P2, and P3 into one. In the following, we discuss various aspects, e. g., what are the differences between the prototypes, what we can learn from the implementation, and which next steps could be taken. Since all three prototypes use RVL as the mapping language, the RVL mapping model as well as the AVM may be stored as a file to continue the processing in one of the other prototypes.

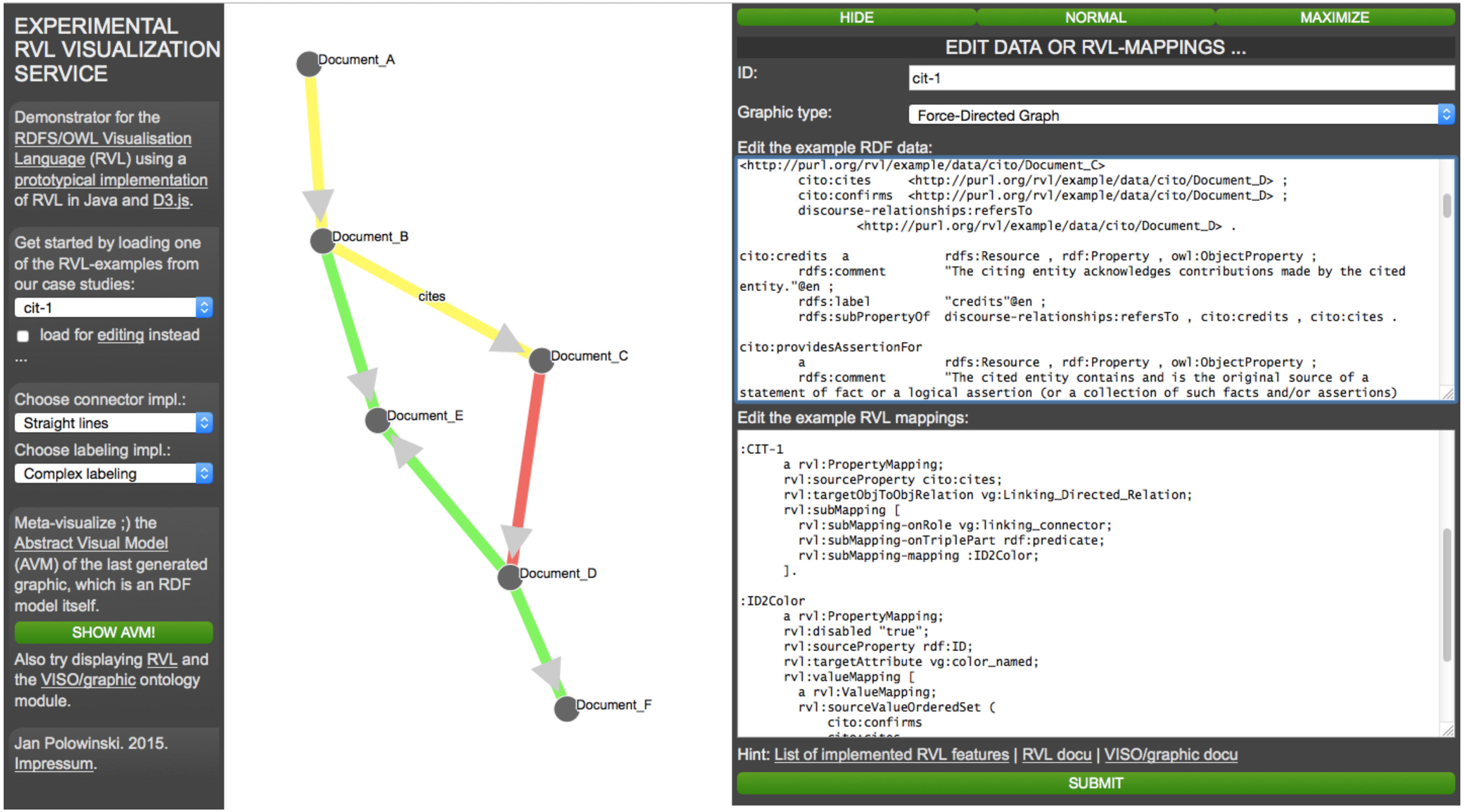
8.6.1 Checking RVL Constraints and Visualisation Rules

The SPIN rules and constraints used in P1 have been reused in P2 (employing the freely available SPIN API) and could easily be reused in P3 and future applications. Even if SPIN does not evolve into a standard, the encapsulated SPARQL queries can still be reused.

8.6.2 A User Interface for Editing RVL Mappings

Fig. 8.12 shows the current web-based UI of the P3 *(Java)* prototype. While the other two prototypes offer modelling the RVL with a generic graphical editor, P3 currently does not. Both data and RVL mappings have to be entered as text and can only be edited in the textual (Turtle)

[5] Apache Jena. http://jena.apache.org/, accessed: 12.2.2016.



EXPERIMENTAL RVL VISUALIZATION SERVICE

Demonstrator for the RDFS/OWL Visualisation Language (RVL) using a prototypical implementation of RVL in Java and D3.js.

Get started by loading one of the RVL-examples from our case studies:

cit-1

□ load for editing instead

...

Choose connector impl.: Straight lines

Choose labeling impl.: Complex labeling

Meta-visualize ;) the Abstract Visual Model (AVM) of the last generated graphic, which is an RDF model itself.

Also try displaying RVL and the VISO/graphic ontology module.

Jan Polowinski. 2015. Impressum.

EDIT DATA OR RVL-MAPPINGS ...

ID: cit-1

Graphic type: Force-Directed Graph

Edit the example RDF data:

```
<http://purl.org/rvl/example/data/cito/Document_C>
        cito:cites     <http://purl.org/rvl/example/data/cito/Document_D> ;
        cito:confirms  <http://purl.org/rvl/example/data/cito/Document_D> ;
        discourse-relationships:refersTo
                <http://purl.org/rvl/example/data/cito/Document_D> .

cito:credits  a            rdfs:Resource , rdf:Property , owl:ObjectProperty ;
        rdfs:comment       "The citing entity acknowledges contributions made by the cited
entity."@en ;
        rdfs:label         "credits"@en ;
        rdfs:subPropertyOf  discourse-relationships:refersTo , cito:credits , cito:cites .

cito:providesAssertionFor
        a                  rdfs:Resource , rdf:Property , owl:ObjectProperty ;
        rdfs:comment       "The cited entity contains and is the original source of a
statement of fact or a logical assertion (or a collection of such facts and/or assertions)
```

Edit the example RVL mappings:

```
:CIT-1
      a rvl:PropertyMapping;
      rvl:sourceProperty cito:cites;
      rvl:targetObjToObjRelation vg:Linking_Directed_Relation;
      rvl:subMapping [
        rvl:subMapping-onRole vg:linking_connector;
        rvl:subMapping-onTriplePart rdf:predicate;
        rvl:subMapping-mapping :ID2Color;
      ].

:ID2Color
      a rvl:PropertyMapping;
      rvl:disabled "true";
      rvl:sourceProperty rdf:ID;
      rvl:targetAttribute vg:color_named;
      rvl:valueMapping [
        a rvl:ValueMapping;
        rvl:sourceValueOrderedSet (
            cito:confirms
```

Hint: List of implemented RVL features | RVL docu | VISO/graphic docu

Figure 8.12: Prototype P3 *(Java)* – Web user interface for experimenting with RVL: On the left, stored example graphics can be selected for display or editing. For each displayed graphic (centre column), the AVM can be shown for demonstration purposes. On the right, a rudimentary editor is provided, mainly consisting of two text fields for editing RVL mappings and data. Changes in data or mappings can be submitted to update the graphic accordingly.

```
1  INSERT {
      GRAPH <http://example.org/avm/graph/> {
         ?go ?tgr ?tv .
      }
   }
6  WHERE {
      GRAPH <http://example.org/avm/graph/> {
         ?go avm:represents ?this .
      }
      GRAPH <http://example.org/rvl/graph/> {
11       ?pm a rvl:IdentityMapping .
         ?pm rvl:sourceProperty ?sp .
         ?pm rvl:targetGraphicRelation ?tgr .
      }
      GRAPH <http://example.org/source/data/graph/> {
16       ?this ?sp ?sv .
         FILTER isLiteral(?sv)
         BIND (?sv AS ?tv)
      }
   }
```

Listing 8.8: Processing an RVL Identity Mapping with a SPIN-rule, which wraps a SPARQL 1.1 Update request to manipulate an RDF graph. The rule expects that a graphic object representing the resource has already been created (Line 8). Also, for clarity, parts of the rule have been removed that clean up obsolete statements in the AVM model when running the rule multiple times.

representation. A (non-generic) graphical editor for RVL could look like the mockup in Fig. 8.14 (editing of submappings is not shown). Furthermore, while we describe, how a GUI for guiding the user during the visual mapping process could be derived, this is not fully implemented by the current prototypes. Only warnings and errors are shown. To avoid unfavourable settings in advance, the GUI should *adapt* to syntactic and perceptual constraints (Sect. 5.7.3) in terms of *suggesting and sorting values* and *preventing* constraint violations (cf. Sect. 8.1.2).

As an alternative to the development of a completely new RVL editor, one could try to further customise OntoWiki to turn it into a more specific editor for RVL mappings or – at least – reuse the constraint checker developed for P2. While the constraints underlying the guided editor from P1 could be reused as well, the generic editor of P1 itself cannot easily be developed into a full specific visualisation suite. Since TopBraid Composer is not open-source, only some modification are possible based on a plug-in mechanism.

8.6.3 Graph Transformations with SPIN and SPARQL 1.1 Update

In P1, we directly worked on the various OGVIC RDF graphs with graph transformations that we defined using SPIN-rules and SPARQL 1.1 Update requests (in Sect. 8.3 we already gave a first example of a graph transformation with SPIN-rules). In P3, the transformations are so far completely done by manipulating the graphs indirectly via Java POJOs (generated with RDFReactor; Sect. 8.5). While some mapping cases cannot conveniently be handled by graph transformations – such as complex mappings of value intervals, others are well suited for the description by graph transformations. Therefore, we expect that a future implementation of RVLInterpreter could benefit from a hybrid approach, using Java code and graph transformations in combination. The use of graph transformations in the context of visual languages has been discussed by Bardohl et. al [BTMS99]. Graph transformations on RDF models with and without SPARQL have been discussed by Braatz and Brandt [BB10].

Listing 8.8 shows another graph transformation, demonstrating how RVL Identity Mappings can be processed with SPIN. A SPARQL INSERT operation adds the resulting statements to the graph containing the AVM model. The listing contains only a few lines, which suggests that using graph transformations could improve the conciseness and elegance of an RVL implementation. To properly decide on a hybrid approach, the following aspects should be examined:

1. Clarity, elegance, conciseness – How precisely can transformations be specified?
2. Evolvability – How flexible can transformations be specified with respect to changes in the VISO, domain data or even RVL?
3. Reusability and documentation – How well can parts of the implementation be documented and reused?
4. Computational universality – Can all value mappings be calculated or do they have to be specified in operational style (e. g., in Java)?
5. Performance – Which transformations are faster when realised as graph transformations with SPARQL Update?

8.6.4 Selection and Filtering of Data

The current prototypes do not support the data selection and filtering steps, which are essential for real-world scenarios. Hence, connecting to existing (web) data sources and querying these sources to select data would be a further necessary step. Also with respect to an end-user evaluation, a filtering component such as *faceted browsing* should be integrated with the system to simplify the reduction of large data sets to the items of interest[6]. Work on the filtering of RDF data has been described in detail by many other approaches (cf. Sect. 4.4.2). It is not covered by the prototypes, except OntoWiki offers basic filtering support and the prototype P2 *(OntoWiki-based)* allows for filtering resources by type.

8.6.5 Interactivity and Incremental Processing

The integration of information filtering into visualisation tools is recommended in order to avoid task switches – at the same time there is a demand for presenting visualisation results »in a just-in-time manner« [Bul08]. So far, our requirements for the OGVIC approach do only cover interactivity on the view modifications level (Sect. 4.1.2) to allow for simple interactions. If we want to extend interactivity to all steps of the visualisation process, i. e., to data selection, filtering and the visual mapping, we need to review if and how short interaction times may conflict with the model-driven, transformation-based approach we suggest. While many general-purpose languages are not made for interaction [LCP$^+$10], creating an interactive architecture using graph transformations on various models may be even more challenging. Related work in the context of behavioural models and animated visual languages [SMPV10] suggests that this may still be feasible.

One option to overcome the long processing times that may result from the transformation chains in a model-driven architecture is to try and implement the incremental processing of only those parts that are effected by the changes. Another option also chosen by Bull [Bul08] in his approach to Model Driven Visualisation is to use a Model-View-Controller (MVC) architecture, partly generating Models, Views and Controllers. Also for future versions of Cytoscape a MVC architecture was considered[7], distinguishing between models, views and view-models.

Fig. 8.13 shows how a MVC architecture for the OGVIC approach may look like. The following principle could be applied: Filtering and mapping views modify the data and mapping

[6] Some filtering is possible already using the RVL subject and object filters, but these are not intended for filtering data, but for refining the applicability of mappings.

[7] http://web.archive.org/web/20110807204359/http://cytoscape.wodaklab.org/wiki/Outdated_Cytoscape_3.0, accessed: 07.08.2011.

models via the controller. For each registered change in the models, the controller triggers the interpretation of the RVL mappings and applies them to the current (filtered) data model. Then all views are notified to trigger the processing of the updated AVM. In more detail, the OGVIC.js controller calls the generator to process the AVM to the format required by the D3.js graphics (JSON) and starts the update mechanism of D3.js, as already implemented in prototype P3. View transformations, like changes to the perspective, are handled by the OGVIC.js controller and do not cause changes to the underlying models, but may affect other views. Changes in the filtering will require the mapping view to refresh, if different mappings and example values are suggested depending on the data. Since the draft of an MVC architecture as suggested above cannot fully be aligned with the *Reference Model* pattern (Sect. 2.1.4) – also a variant of MVC for visualisation – both approaches should be carefully compared. Differences emerge from the fact that we describe multiple views (we add a filtering and editing view), but also from the fact that we distinguish between the mapping model and the AVM. Prototype P3 *(Java)* already partly implements a MVC architecture, e. g., the actual data graph is kept separately from the view implemented with D3 and its view model.

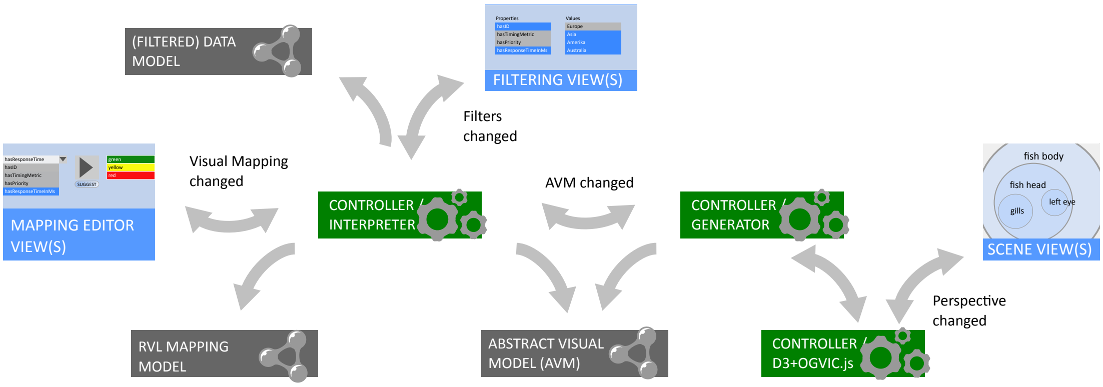
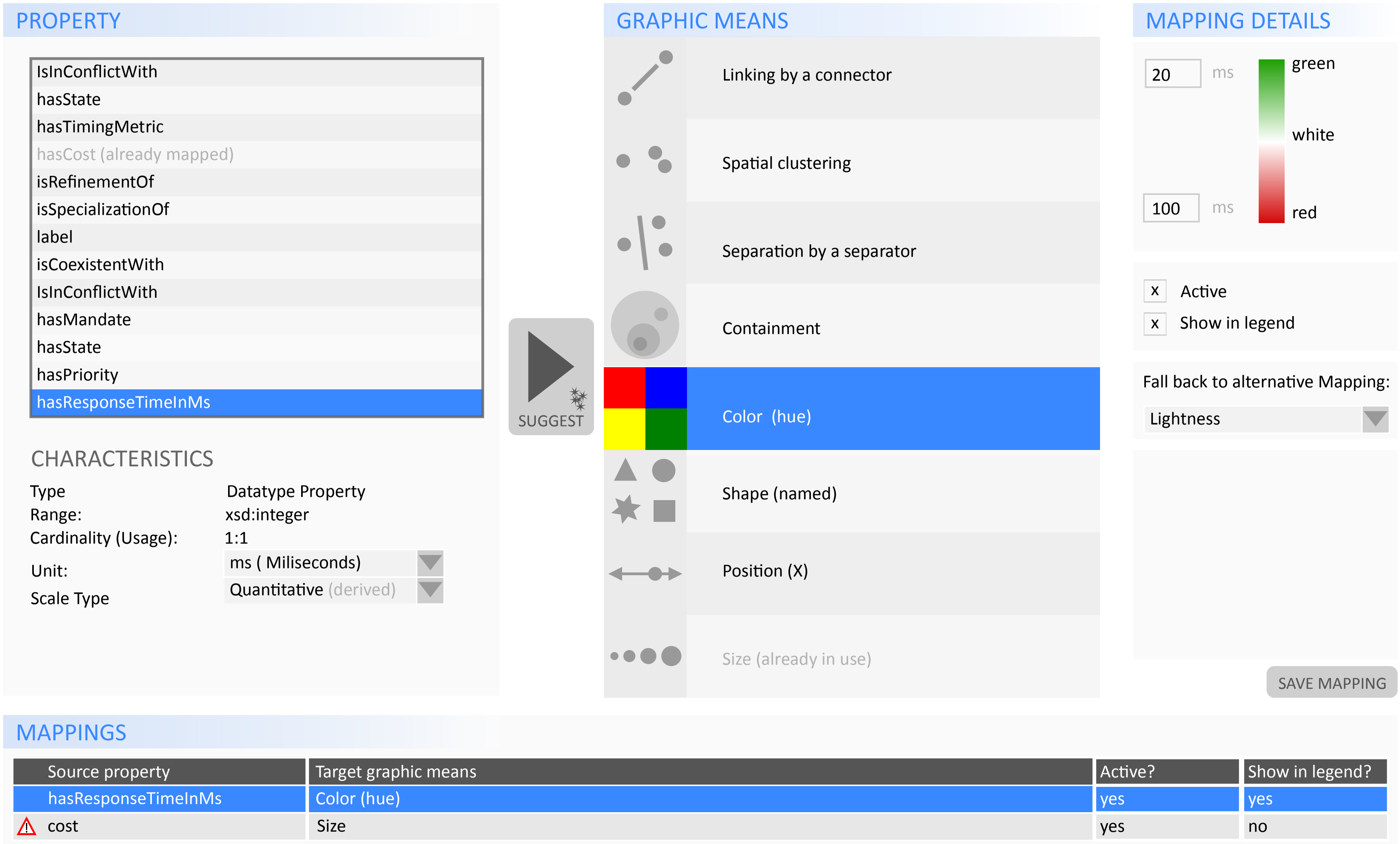


Figure 8.13: Simplified representation (no distinction between control and data flow) of how the OGVIC approach could be realised as a Model-View-Controller architecture.



PROPERTY

- IsInConflictWith
- hasState
- hasTimingMetric
- hasCost (already mapped)
- isRefinementOf
- isSpecializationOf
- label
- isCoexistentWith
- IsInConflictWith
- hasMandate
- hasState
- hasPriority
- hasResponseTimeInMs

CHARACTERISTICS

Type	Datatype Property
Range:	xsd:integer
Cardinality (Usage):	1:1
Unit:	ms (Miliseconds)
Scale Type	Quantitative (derived)

SUGGEST

GRAPHIC MEANS

- Linking by a connector
- Spatial clustering
- Separation by a separator
- Containment
- Color (hue)
- Shape (named)
- Position (X)
- Size (already in use)

MAPPING DETAILS

20 ms — green

white

100 ms — red

☒ Active

☒ Show in legend

Fall back to alternative Mapping: Lightness

SAVE MAPPING

MAPPINGS

Source property	Target graphic means	Active?	Show in legend?
hasResponseTimeInMs	Color (hue)	yes	yes
⚠ cost	Size	yes	no

Figure 8.14: Mockup of a user interface for an RVL mapping editor. Visualisation authors can create and edit mappings by selecting a property from the left and a graphic relation from the right list. Clicking the *SUGGEST* button automatically picks the most effective, not yet used graphic relation for the selected property. These settings can be manually changed and refined by defining value mappings in the *Mapping details* area. At the bottom of the screen a list of already created mappings is shown. A warning sign is depicted next to non-optimal mappings. Further things to notice are the option to manually correct the derived scale type and unit (here *quantitative* and *milliseconds*). The handling of submappings is not shown.

8.6.6 Rendering the Final Platform-Specific Code

Unlike the SPIN–constraints and -rules, the rendering implemented with UISPIN for prototype P1, could not be transferred and reused in the other prototypes. Although not only an HTML, but also a SVG vocabulary for UISPIN is available, neither of them is capable of generating data-driven interactive graphics, as possible with D3. Furthermore, we would have had to write our own UISPIN interpreters, since there is no open-source implementation by now. For these reasons, in P2 and P3 we take the alternative approach of generating[8] a JSON representation of the AVM, which can then be used to drive D3 graphics.

In Listing 8.7, we already gave an example of the JSON format that we use in prototype P3. In its current state it cannot yet express the complete AVM in a uniform way. Instead we provide two variants of the AVM as JSON, one for graphic types expecting a flat structure (such as *Force-directed graph*), and one for those expecting a hierarchical data structure (such as *Collapsible tree* and *Zoomable circle packing*, cf. Listing D.3 in the appendix). To simplify processing on the client side, we sometimes explicitly add information that is only implicitly contained in the AVM (such as the type of labels). Future work on the AVM should include defining a single »AVM in JSON« format that can be processed by all OGVIC D3 plugins. Instead of generating multiple variants on the server side, it may be considered to transform into a hierarchical representation on the client side (with JavaScript).

Additionally, it is currently required that a graphic type and layout is provided as part of the JSON object. In later implementations, the system should assume a default graphic type and layout (based on the graphic relations used in the AVM) so that this information becomes optional.

One might argue that determining a concrete graphic type at all is against the principle of OGVIC, since we claimed to focus on (composable) graphic relations instead of fix graphic types in order to achieve a high degree of flexibility. This is true, and a solution without fix graphic types is clearly preferable. While we already work directly on graphic relations on the level of the AVM, the D3 interpretation in prototype P3 is currently built around predefined graphic types such as *Force-Directed Graph*[9], *Collapsible Tree*[10] or *Zoomable Circle Packing*[11]. However, also on the level of rendering, the use of D3 offers a good foundation to work directly on graphic relations, since D3 is a flexible language rather than a visualisation toolkit. Already now, we use »D3 plugins« (modifications of the *selection* prototype) to uniformly apply graphic attributes and simple interactions across graphic types. We, therefore, consider building a single (D3-based) interpreter for arbitrary AVM models a promising next step.

[8] An alternative to writing the JSON with Java, as done in P2 and P3, could be to use SWON, a UISPIN library for JSON. http://uispin.org/swon.html, accessed: 07.01.2016.

[9] Based on an example from http://bl.ocks.org/mbostock/4062045/, accessed: 29.04.2016

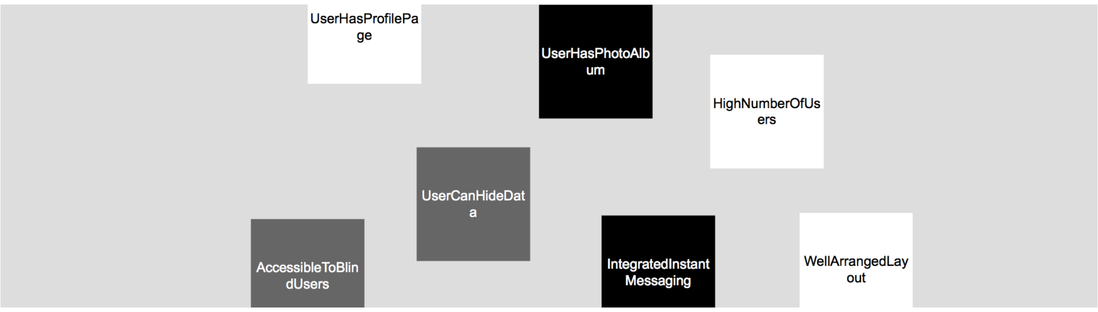
[10] Based on an example from http://bl.ocks.org/mbostock/4339083/, accessed: 29.04.2016

[11] Based on an example from http://bl.ocks.org/mbostock/7607535/, accessed: 29.04.2016

Chapter 9

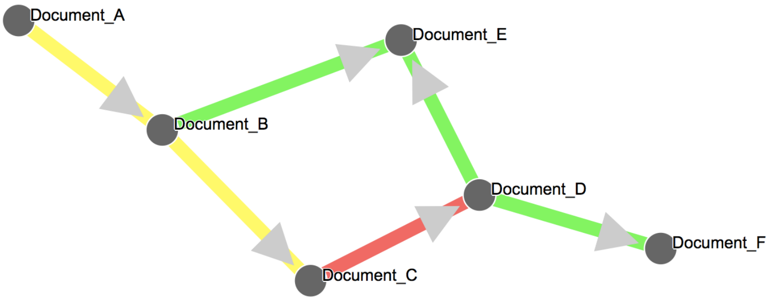
Applying the OGVIC Prototypes to the Case Studies' Ontologies

An evaluation of the general usefulness of the OGVIC approach for arbitrary domains is out of scope of this thesis. Instead, in this chapter, we do a constructive evaluation and challenge both the RVL language specification and the prototypical implementations of our approach by trying to generate the sketches that we manually created during the requirements analysis (Sect. 9.1). In a second and third step, we continue the coverage analysis for visualisation cases (Sect. 9.2) and general requirements (Sect. 9.3). Finally, we give a full example of a guided visualisation process using the OGVIC prototypes (Sect. 9.4).



```
:RO-4b
  a rvl:PropertyMapping;
  rvl:sourceProperty rdf:type;
  rvl:targetAttribute vg:color_hsl_lightness;
  rvl:valueMapping [
       rvl:sourceValueOrderedSet (
             ro:LowPriority
             ro:MediumPriority
             ro:HighPriority
       );
       rvl:targetValueInterval [
       rvl:lowerBoundIncl "0";
       rvl:upperBoundIncl "100";
   ];
  ].
```

Figure 9.1: RO-4b (rdf:type ↦ lightness) – Generated graphic and the corresponding RVL mapping for the example RO-4b from the *Requirements Ont.* (ro:). Priority was modelled as classes in the Requirements Ontology (ro:LowPriority, ro:MediumPriority and ro:HighPriority), so we have to select rdf:type as our source property and add a value mapping. Position (force-driven) and text color are rendering defaults, not defined by RVL.

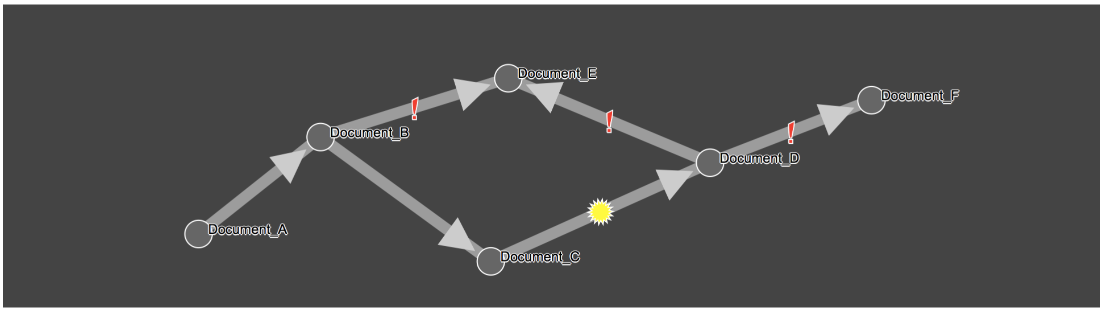


```
:CIT-1
     a rvl:PropertyMapping;
     rvl:sourceProperty cito:cites;
     rvl:targetObjToObjRelation vg:Linking_Directed_Relation;
     rvl:subMapping [
       rvl:subMapping-onRole vg:linking_connector;
       rvl:subMapping-onTriplePart rdf:predicate;
       rvl:subMapping-mapping :ID2Color;
     ].

:ID2Color
     a rvl:PropertyMapping;
     rvl:disabled "true";
     rvl:sourceProperty rdf:ID;
     rvl:targetAttribute vg:color_named;
     rvl:valueMapping [
       a rvl:ValueMapping;
       rvl:sourceValueOrderedSet (
           cito:confirms
           cito:cites
           cito:critiques
       );
       rvl:targetValueList rexc:ColorsTrafficLight;
     ].
```

Figure 9.2: CIT-1 (cito:cites ↦ directed linking + subproperties distinguished by a set of ordered colours (rexc:ColorsTrafficLight)) – Generated graphic and the corresponding RVL mapping for the example CIT-1 from the *Citation Ont.* (cito:).

9.1 Coverage of Sketches and Necessary Features

In the following, we first give a visual overview of the results by showing the generated graphics for some of the sketches we created during the analysis (Fig. 9.1–9.7). Below each screenshot of a rendered graphic, we list the corresponding RVL mapping. We sometimes exclude settings that are only necessary due to the current implementations' limited support of default values or styles (see Appendix D.1 for listings including these settings). The seven listings demonstrate the usage of property mappings to both graphic attributes (colour, shape, size, lightness and text value) and graphic object-to-object relations (linking, containment, labelling, relative distance). Furthermore, manual and automatic value mappings are used to map sets and ranges of source values with various scales of measurement to target graphic values. Some listings are supplemented by CSS style settings (here given as comments below the RVL code), which

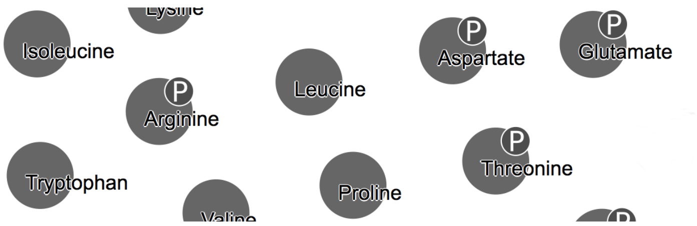


```
:CIT-5
     a rvl:PropertyMapping;
     rvl:sourceProperty cito:cites;
     rvl:targetObjToObjRelation vg:Linking_Directed_Relation;
     rvl:subMapping [
       rvl:subMapping-onRole vg:linking_connector;
       rvl:subMapping-onTriplePart rdf:predicate;
       rvl:subMapping-mapping :IconLabelMapping;
     ].

:IconLabelMapping
     a rvl:PropertyMapping ;
     rvl:disabled "true";
     rvl:sourceProperty rdf:ID;
     rvl:targetObjToObjRelation vg:Labeling_Relation;
     rvl:subMapping [
       rvl:subMapping-onRole vg:labeling_label;
       rvl:subMapping-onTriplePart rdf:predicate;
       rvl:subMapping-mapping :IconLabelShapeMapping;
     ].

:IconLabelShapeMapping
     a rvl:PropertyMapping;
     rvl:disabled "true";
     rvl:sourceProperty rdf:ID;
     rvl:targetAttribute vg:shape_named;
     rvl:valueMapping [
       rvl:sourceValueOrderedSet (
         cito:critiques cito:confirms );
       rvl:targetValueList (
         common-shapes:XMark common-shapes:Star18 );
     ].
```

Figure 9.3: CIT-5 (cito:cites ↦ linking + iconic labels on the connector to distinguish subproperties of cites) – Generated graphic and the corresponding RVL mapping for the example CIT-5 from the *Citation Ont.* (cito:). The ordered set of subproperties is mapped to a list of shapes: cito:critiques ↦ common-shapes:XMark (exclamation mark), cito:confirms ↦ common-shapes:Star18 (star-shape).

describe those graphic properties that do not encode meaning and hence are not part of the visual mapping. For example, the default size of graphic objects and the default width of connectors is set using CSS (e. g., AA-3, Fig. 9.4). We comment on each listing in the respective caption in order to provide the explanation close to the RVL code. Therefore, in the following, we only give a very brief overview of the seven listings and relate them to one another. Page forward to compare the listings and figures. For details on the utilised language constructs refer to Chapter 7.

We begin with mapping example RO-4b (Fig. 9.1), a compact example from the software domain, which maps requirements to discrete greyscale values depending on their priority. Although RVL does not introduce its own dedicated syntax, the two mappings (a property mapping and a value mapping) can be concisely represented in Turtle. The second mapping

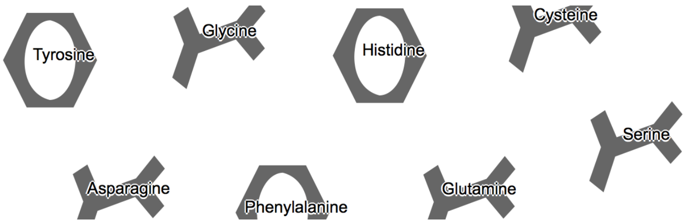


```
:AA-3
     a rvl:PropertyMapping;
     rvl:sourceProperty amino-acid:hasPolarity;
     rvl:objectFilter "http://www.co-ode.org/../amino-acid../Polar^^rvl:classSelector;
     rvl:inheritedBy owl:allValuesFrom;
     rvl:targetObjToObjRelation vg:Labeling_Relation;
     rvl:subMapping [
       rvl:subMapping-onRole vg:labeling_label;
       rvl:subMapping-onTriplePart rdf:object;
       rvl:subMapping-mapping :Polar_to_CircleP;
     ].

:Polar_to_CircleP a rvl:ResourceMapping;
     rvl:disabled "true"; # only active in submappings
     rvl:sourceValue amino-acid:Polar;
     rvl:targetValue common-shapes:CircleP.

# Additional CSS settings for constant values

     #canvas.aa-3 .labeling_base {
       transform:scale(3.0) ;
     }
```

Figure 9.4: AA-3 (aa:hasPolarity ↦ label with a compass shape but only if the value is aa:Polar) – Generated graphic and the corresponding RVL mapping for the example AA-3 from the *Amino Acid Ont.* (aa:). Setting the size of the objects to a higher constant value is done by CSS, since size does not encode meaning in this example.

example, CIT-1 (Fig. 9.2), resembles the submapping example from Sect. 7.10, but is slightly modified to define a specific set of target values replacing the default one. CIT-5 (Fig. 9.3) again modifies the mapping by switching the colour-submapping to a submapping that labels connectors with icons and distinguishes subproperties by icon shape instead of colour. This demonstrates how small variations in an RVL mapping result in a different graphic. CIT-1 and CIT-5 also illustrate how the rvl:submapping_onRole and rvl:submapping_onTriplePart properties of RVL are used to define a context for the submappings. AA-3 (Fig. 9.4) is an example of how to apply mappings only to a filtered set of resources. AA-4 (Fig. 9.5) shows how to extend (rvl:inheritedBy) property mappings to indirectly stated relations that emerge from universal class restrictions in the T-Box of ontologies. The mechanism of extending property mappings avoids additional language constructs and keeps the set of necessary classes small. In PO-9 (Fig. 9.6), we use resource mappings instead of property mappings, since this leads to more compact mapping definitions if only few values need to be addressed. PO-5 (Fig. 9.7) is again a simple property mapping of rdfs:subClassOf to containment. In order to achieve the desired structure (a class contains its subclasses), we need to invert the source property, though.

Table 9.1 gives an overview of how many sketches from our case studies can be described based on the current specifications of RVL, VISO, and the AVM and how many sketches can be implemented with prototype P3. We distinguish between the overall set of sketches we did during the problem analysis and the subset of sketches we selected as examples in Chapter 3. The complete list of sketches is given in Appendix A. To consider also those sketches that are missing some of multiple necessary features, we state to what amount the necessary set of features can be described / has been implemented (100% / at least 75% / at least 50%, and less than 50%).



```
:AA-4
     a rvl:PropertyMapping ;
     rvl:sourceProperty amino-acid:hasSideChainStructure ;
     rvl:inheritedBy owl:allValuesFrom ;
     rvl:targetAttribute vg:shape_named ;
     rvl:valueMapping
     [ rvl:sourceValue amino-acid:Aliphatic ;
       rvl:targetValue bio-shapes:Aliphatic_Shape ],
     [ rvl:sourceValue amino-acid:Aromatic ;
       rvl:targetValue bio-shapes:Aromatic_Shape ].

# Additional CSS settings for constant values

     #canvas.aa-4 .labeling_base {
       transform:scale(3.0) ;
     }
```

Figure 9.5: AA-4 (aa:hasSideChainStructure ↦ shape) – Generated graphic and the corresponding RVL mapping for the example AA-4 from the *Amino Acid Ont.* (aa:). Setting the size of the objects to a higher constant value is done by CSS, since size does not encode meaning in this example.

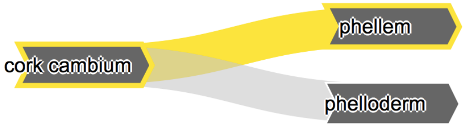
A large amount, but not all sketches from our three case studies can be described based on the current RVL/VISO/AVM specification. The overall set of sketches consists of 37 sketches. Over two thirds of them (26 sketches) can be almost completely specified (at least by 75%); another 5 sketches can be specified by at least 50%. With respect to the technical implementation with our prototypes, prototype P3 is the one that covers most of the RVL specification. About half of the sketches (19), we can implement with P3 by at least 75%; another 8 sketches can be implemented by at least 50%. The main reason why sketches could not yet be described, is the missing support for the complex graphic relation types used in these sketches (e. g., *adjacency* and *builds-on*; especially ZFO-3x and ZFO-5). For PO-5, *containment* needs to be implemented for arbitrary shapes.

For the subset of characteristic sketches we focused on in Chapter 3, the coverage is a little higher. The subset consists of 12 sketches, of which 10 can be almost completely specified (at least by 75%). AA-5 and PO-9 require further conceptual work – for instance, we need to decide on a concept for describing tables and flow-diagram-like connectors. Except for three sketches (AA-5, PO-9, RO-7), we can also implement them with P3 by at least 75%.

This can only be a rough estimation of the coverage, since we do not weight features and features are used in multiple sketches. A list of the features that are missing to completely reproduce the subset of sketches from Chapter 3 can be found in Appendix D.2. In Table D.1, we list the total set of features that we identified as necessary to recreate all sketches. From these features, 53% are covered by the current specifications of RVL, VISO, and the AVM. Additional 9% can partly be specified, but require further conceptual work. 57% of the sketches are implemented by prototype P3 and additional 4% are partly implemented.

9.2 Coverage of Visualisation Cases

We proceed with the coverage analysis on the level of visualisation cases (VC) that we identified in Chapter 3. As listed in Sect. 7.12, some VC are not yet covered by the specification of RVL. In continuation of the comparison, Table 9.2 provides an overview of which prototype (P1, P2



```
:PO-9
     a rvl:PropertyMapping;
     rvl:sourceProperty obo:develops_from;
     rvl:invertSourceProperty "true";
     rvl:passedTo owl:someValuesFrom;
     rvl:targetObjToObjRelation vg:Linking_Directed_Relation;
     rvl:subMapping [
       rvl:subMapping-onRole vg:linking_connector;
       rvl:subMapping-onTriplePart rdf:predicate;
       rvl:subMapping-mapping :ArrowShapeMapping;
     ].

:ShapeMapping a rvl:ResourceMapping;
     rvl:sourceValue <http://purl.org/obo/owl/PO#PO_0025059>;
     rvl:targetValue common-shapes:FlowArrow;
     rvl:passedTo rdfs:subClassOf.

:ArrowShapeMapping a rvl:ResourceMapping;
     rvl:disabled "true";
     rvl:sourceValue obo:develops_from;
     rvl:targetValue common-shapes:Line.

# Additional CSS settings for constant values

     #canvas.po-9 .linking_connector {
       stroke-width: 23;
     }
```

Figure 9.6: PO-9 (po:develops_from^{-1} ↦ linking (with specially shaped connectors) + highlighting of direct neigbors when hovering) – Generated graphic and the corresponding RVL mapping for the example PO-9 from the *Plant Ont.* (po:). Setting the width of the connector to a higher constant value is done by CSS, since it does not encode meaning in this example.

or P3) implements a given visualisation case. (x) means that the visualisation case is basically fulfiled, but some aspects are missing. (-) means the visualisation case is basically not supported, with some exceptions.

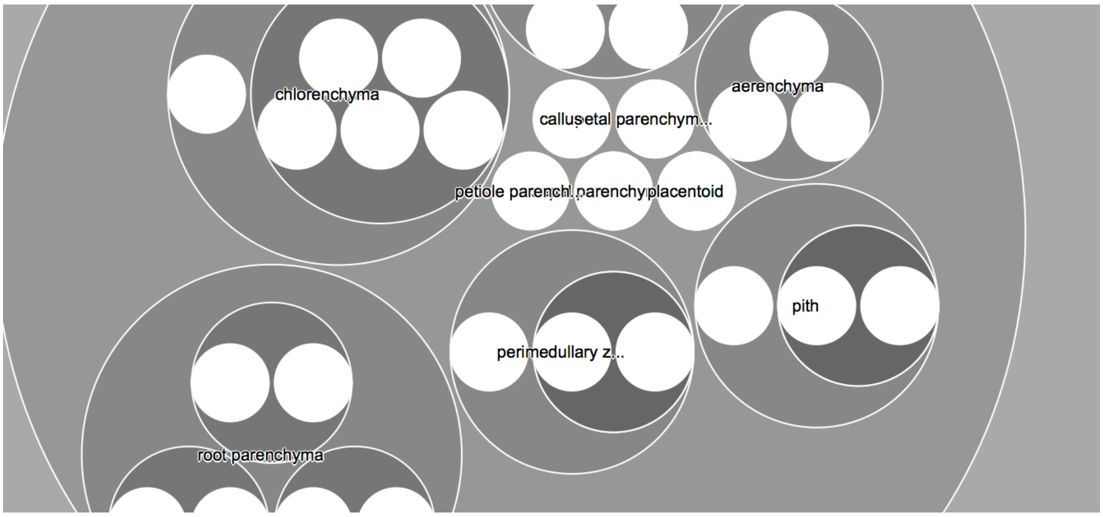
All prototypes support the creation of a graphic object per resource *(VC-1)* and the mapping to graphic attributes *(VC-2)*. The latter is only supported in a basic form, since rendering is implemented for a subset of attributes, not all attributes defined in VISO. Also the mapping to graphic object-to-object relations *(VC-3)* is implemented in P1 and P2 for *containment* and *linking*. P3 differentiates between *undirected* and *directed linking* and adds *labelling* as well as *relative distance* (clustering). As required for labelling, also the creation of additional graphic objects – not directly representing a resource – is supported, but not for arbitrary graphic relations *(VC-4)*. Some simple interactions *(VC-5)* are implemented in P1 and P3, however, these interactions are currently applied by default in P3 and need to be (de)activated according to the RVL settings. The simplification mechanisms *(VC-6)* suggested as part of RVL have not yet been in the focus of implementation. Reuse and composition *(VC-7)* are supported with P3. However, P3 supports only a subset of the composition cases that we described and the applicability of compositions with respect to the underlying data cannot yet be determined with P3. As a prerequisite for composition, P3 also supports referring to parts of the graphic *(VC-9)* by the role of a graphic object. Standard graphics *(VC-8)* are not yet covered by the RVL specification, but experimentally implemented in the prototypes P2 and

set of sketches	all sketches		subset from Sect. 3.6	
number of sketches	37		12	
	described	implemented	described	implemented
completely (100%)	21	10	6	3
by [75%, 100%)	5	9	4	6
by [50%, 75%)	5	8	1	2
by less than 50%	6	10	1	1

Table 9.1: Number of sketches that can be implemented with prototype P3, respectively described based on the current specification of RVL, VISO and AVM. The subset of sketches refers to the selection of sketches that we introduced in Sect. 3.6. Where the number of implemented sketches is higher than those described, this is due to the fact that some features are covered by default settings implemented in P3 that did not yet become part of the specification of RVL, VISO or the AVM or are not intended to become part of these specifications.

		RVL	P1 (»TopBraid Composer«)	P2 (»OntoWiki«)	P3 (»Java«)
VC–1	Create a graphic object per resource	x	x	x	x
VC–2	Map to graphic attributes	x	(x)	(x)	(x)
VC–3	Map to graphic object-to-object relations	x	(-)	(-)	(x)
VC–4	Create additional graphic objects	(x)	(-)	(-)	(x)
VC–5	Define simple interactions	(x)	(x)	(-)	(x)
VC–6	Simplify the ontological model	x	(x)	-	-
VC–7	Reuse/extend/compose mappings	(x)	(-)	(-)	(x)
VC–8	Use complex standard graphics	-	-	(x)	(x)
VC–9	Refer to parts of the graphic	x	-	-	x
VC–10	Draw legends and labelled axes	(x)	-	(x)	-
VC–11	Define styles	-	(-)	(-)	(x)
VC–12	Benefit from good defaults	x	-	-	(x)

Table 9.2: Visualisation cases covered by the prototypes. Additionally, for comparison, the coverage of RVL is repeated.



```
:PO-5
     a rvl:PropertyMapping;
     rvl:sourceProperty rdfs:subClassOf;
     rvl:invertSourceProperty "true";
     rvl:targetObjToObjRelation vg:Containment_Relation.
```

Figure 9.7: PO-5 (rdfs:subClassOf^{-1} ↦ containment) – Generated graphic and the corresponding RVL mapping for the example PO-5 from the *Plant Ont.* (po:). Labels abbreviated with »...«.

P3. Drawing legends *(VC-10)* is explicitly supported by the rvl:includeInLegend flag, but only implemented by P2. Deriving legends from RVL mappings seems to be straightforward, though. The labelling of axes is not yet covered. Styling as a complement of visual mapping *(VC-11)* did not become part of RVL. Instead Fresnel and CSS are intended to be reused for this purpose.

While none of the prototypes supports Fresnel, global CSS styles such as the default colour of paths and texts can be used in all prototypes. Since SVG and HTML are the final platforms targeted by the prototypes, CSS styles can simply be interpreted by the browser and are overridden by local style values generated from the visual mappings where necessary. P3 automatically assigns CSS classes based on the role that graphic objects play, e. g., *.linking_connector*. These classes can be used to define a style for all graphic objects with a specific role. For example, in mapping *PO-9* (Fig. 9.6) the *stroke-width* is defined for all connectors using CSS (here the style rule is additionally limited to the generated graphic with the ID *po-9*):

```
canvas.po-9 .linking_connector {
   stroke-width: 23;
}
```

Also other attributes, such as the size of graphic objects can be defined by styles using *transform:scale()*. However, *shape* cannot be set by CSS, since it is not considered an attribute. As a workaround for these cases, we currently have to assign a constant value with an RVL mapping (e. g., the :ShapeMapping in mapping *PO-9*).

Some of the defaults *(VC-12)* specified in RVL are implemented by P3, for example, the default labelling of graphic objects, default graphic attribute value sets, and the defaults assumed during the process of determining the source values for a mapping. An important default that lacks implementation, though, are default value mappings (Sect. 7.7) that would allow for defining property mappings without dealing with the details of explicit value mappings. Furthermore, P3 implements defaults that are not specified in RVL, but rather concern the styling. For instance, text colour is always set to a value that is readable on the given background colour (unless the text colour encodes meaning). More defaults are hardcoded to define a concrete rendering for directed connectors (↦ arrows) and undirected connectors (↦ lines). Some defaults, such as

		P1 (»TopBraid Comp.«)	P2 (»OntoWiki«)	P3 (»Java«)	Approach
R-1	Dynamic and value-dependent visual mapping	(x)	x	x	x
R-2	Variety of graphic relations	-	(-)	(x)	x
R-3	Interaction with the visualisation	(x)	(-)	(x)	x
R-4	Ontology-aware (A/T-Box)	x	(x)	x	x
R-4a	Terminological ontology relations (T-Box) supported	(x)	-	x	x
R-5	RDF supported	x	x	x	x
R-6	Domain agnostic	x	(x)	x	x
R-7	Reusability of the defined mappings	(x)	(x)	x	x
R-8	Composability of the defined mappings	(-)	(-)	(x)	(x)
R-9	Explicit mapping definitions	x	x	x	x
R-10	Platform variability	-	-	(x)	x
R-11	Visual structure variability	-	-	x	x
R-12	Domain experts can visualise their data without progr. or vis. skills	(-)	(-)	-	x
R-13	Visualisation settings configurable with a GUI	(x)	x	(-)	x
R-14	Interactions configurable with a GUI	(x)	-	-	x
R-15	Guidance for visual mapping with a GUI	(x)	(x)	-	x
R-16	Consider complex semantics of an ontology for visual mapping	(x)	(-)	-	x
O-1	Configuration results instantly shown	-	(x)	(x)	(x)
O-2	Data filtering	-	(x)	-	(x)
O-3	Guidance for data selection	-	-	-	-
O-4	Guidance for view transformations	-	-	-	-

Table 9.3: Requirements and optional features covered by the prototypes.

the colour used for highlighting, can already be specified using CSS. For others, not covered by CSS, it has to be investigated how they could be defined in a similarly flexible way.

While we defined the creation of a graphic object per resource *(VC-1)* as our first visualisation case, it turned out that there are also many cases where we do not want this behaviour. Exceptions to this general principle can be handled with the submapping mechanism, since submappings allow for constraining the application of mappings to a specific context.

In summary, prototype P3 covers ten of twelve visualisation cases, like the RVL specification. However these cases are not the same covered by the specification. Some are covered by specification while not being implemented or only implemented with restrictions. Some cases are implemented, but not yet described in the RVL specification. The visualisation cases not covered by P3 are partly covered by the other two prototypes.

9.3 Coverage of Requirements

As the last part of the coverage analysis, let us recall the requirements we set. Table 9.3 gives an overview of which prototype meets which requirement. The OGVIC approach (last column) meets all requirements, except for some limitations concerning the generalisation of the composition of graphic relations (Sect. 6.6). Compositions, such as the construction of a tabular representation from graphic relations, require further conceptualisation and testing. We experimented with multiple variants of expressing table structures with the AVM, but did not decide on a recommended approach. With respect to the optional requirements, the instant feedback for configuration results *(O-1)* and data filtering *(O-2)* have been briefly discussed. No conceptual work was done in the field of guidance for data selection *(O-3)* and view transformations *(O-4)*.

To summarise the coverage by the prototypes, we focus on the last prototype P3 and only refer to P1 and P2 where requirements are not met. P3 interprets visual mappings *(R-1)* defined with RVL for a number of graphic relations *(R-2)*, though not for all the graphic relations defined in the VISO ontology. Interactivity *(R-3)* is so far provided on the level of view transformations, e. g., for highlighting graphic elements. No interactions have yet been implemented on the level of visual mapping and data filtering. Prototype P3 is aware of ontologies *(R-4)*, including that it can handle T-Box relations *(R-4a)*. Thereby, the implementation is specific to the RDF technical space *(R-5)* but agnostic to the domain of knowledge *(R-6)*, i. e., it can be used to visualise any RDF-based ontologies and instance data. Since P3 can handle the concept of submappings, reuse *(R-7)* and composability *(R-8)* of mappings are generally supported. However, as already mentioned with respect to the visualisation case coverage, work has to be done to cover a larger amount of possible composition cases. Some compositions of graphic relations, such as the composition of *linking* and *containment*, can be easily described with the AVM, but require a more complex rendering and layouting than we currently implemented in our prototypes. Since not all graphic relations can be freely composed without paying attention to syntactical and perceptual constraints (Sect. 5.7.3), additional rules have to be created and evaluated during the mapping process. This is not accomplished by the current prototypes. The explicit definition of mappings *(R-9)* and the AVM build the foundation for the support of variability. Switching to another platform *(R-10)*, e. g., from a D3-based SVG rendering to X3D output, is supported by the current architecture, but only the rendering to SVG is implemented. Visual structures emerge from the combination of various graphic relations, which allows for a high degree of variability *(R-11)*. While P3 is the prototype that interprets the largest fraction of the RVL specification, it lacks user interface and guidance features to actually enable the visualisation by domain experts without programming or visualisation skills *(R-12)*. Here the other prototypes, P1 and P2, offer more functionality: Both the TopBraid-Composer-based and the OntoWiki-based prototype let the user configure RVL mappings based on the tools' graphical editors *(R-13)*. In the case of TopBraid Composer, it was possible to customise the generic RDF editing UI to be RVL-specific and thereby simplify the handling of mappings including basic interactions *(R-14)*. In the case of OntoWiki, a plugin was built to enable and disable mappings. Similarly, both P1 and P2 offer basic guidance functionality for the visual mapping *(R-15)* as described in Sect. 8.1.3. P1 is also considering the specifics of complex ontologies *(R-16)* for guidance, e. g., by evaluating class restrictions and subproperty hierarchies.

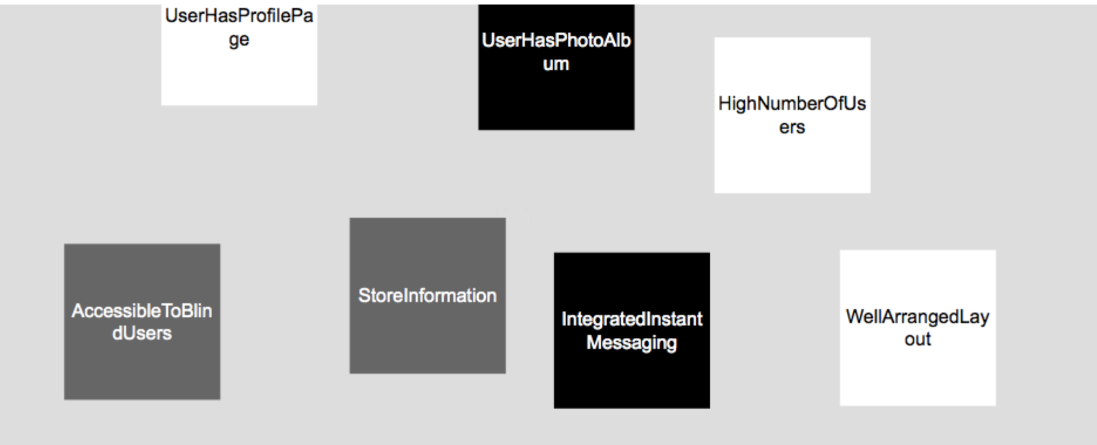
From the optional »requirements«, P2 and P3 already support the instant representation of changes to the visualisation settings *(O-1)*. The advanced update mechanism of D3.js, allows for keeping layout information, while applying (animated) changes to the modified graphic values. Finally, P2 offers generic data filtering based on the OntoWiki infrastructure *(O-2)*.

To summarise, while the OGVIC approach covers almost all requirements (15 out of 17), P3 fulfils about two thirds of them (12) of which four are only partly fulfilled. From the five not fulfilled requirements, four are at least partially covered by prototype P1 or P2. However, none of the prototypes is yet in a state to allow domain experts without programming or visualisation skills to visualise their data.

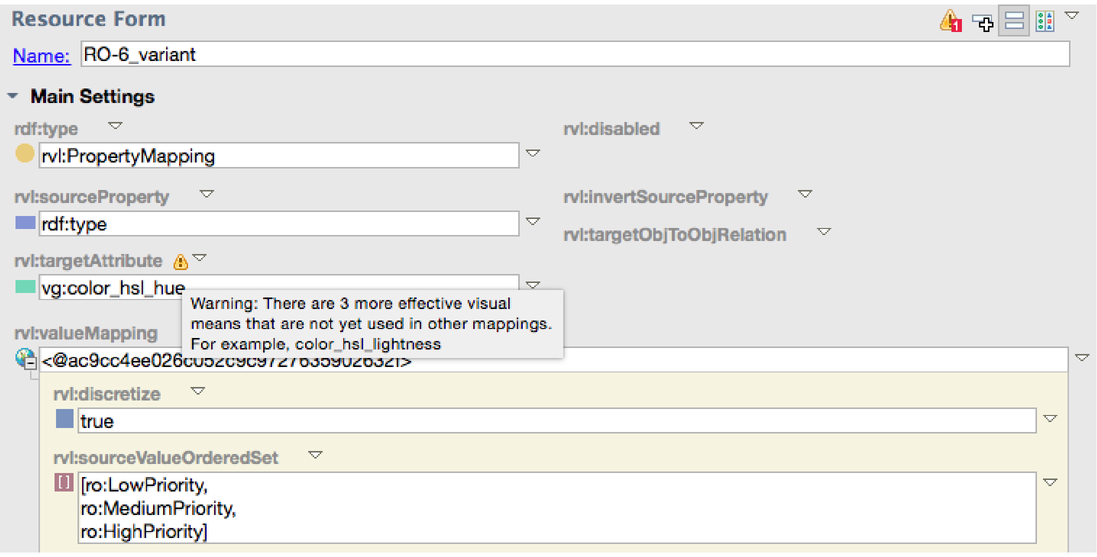
9.4 Full Example

In the following, we revisit the introductory example, which described a guided visualisation process for visualising requirements specified with the *Requirements Ontology* (Fig. 1.3). We demonstrate, to which amount the scenario can already be implemented and where are the weaknesses of the current prototypes. Compared to the larger example we already gave at the end of the RVL chapter, which focused on the submapping mechanism, in this example, we illustrate how a complex mapping can be created step by step. For each intermediate step, we provide a listing of the added mapping or a screenshot of the editing UI as well as the resulting graphic.

Since we are using prototype P1 for editing our mappings, but P3 for interpreting and rendering the AVM, we cannot instantly see the results of our changes, but have to save the mapping files with P1 and reload them in P3. The figure below shows the graphic resulting from an initial mapping from rdf:type[1] to *lightness*; this equals mapping RO-4b, Fig. 9.1.



There is not yet a mechanism to guide the user in a way that important relations and suitable visual mappings would be suggested in a stepwise manner. However, when using P1 to create the mappings, detected non-optimal mapping choices will generate warnings and alternatives will be suggested. For example, if we change the attribute *lightness* to *colour hue*, which is less effective for encoding the ordinal priority values, a warning is generated, telling us that three other more effective graphic attributes are still available (screenshot below). Additionally, a quickfix allows us to remove the non-optimal graphic relation setting (quickfixes could also be extended to *replace* the non-optimal setting with the suggested alternative). Having selected lightness, there is still one more effective attribute – *position*, which we do not select, but reserve for layout purposes, though.


Resource Form

Name: RO-6_variant

Main Settings

rdf:type
rvl:PropertyMapping

rvl:disabled

rvl:sourceProperty
rdf:type

rvl:invertSourceProperty

rvl:targetObjToObjRelation

rvl:targetAttribute
vg:color_hsl_hue

Warning: There are 3 more effective visual means that are not yet used in other mappings. For example, color_hsl_lightness

rvl:valueMapping
<@ac9cc4ee026c052c9c97276359026321>

rvl:discretize
true

rvl:sourceValueOrderedSet
[ro:LowPriority,
ro:MediumPriority,
ro:HighPriority]

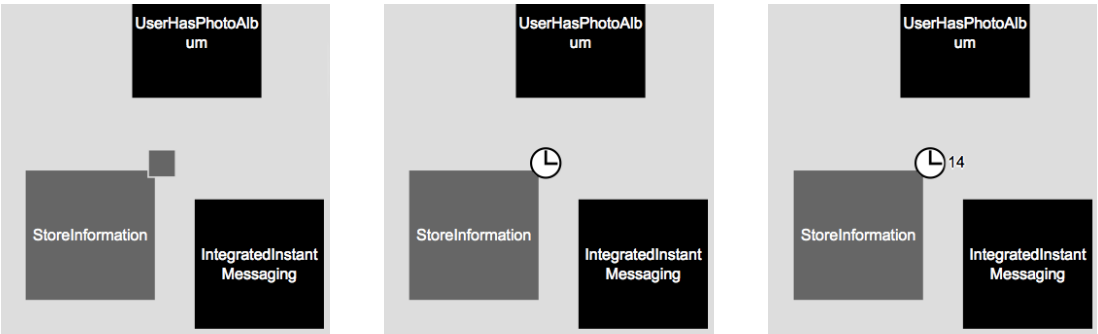
The corresponding SPIN-constraint is similar to the one shown in Listing 8.3. However, in this case, we cannot determine the scale of measurement from the source property directly (stating globally that rdf:type is ordinal obviously makes no sense). The system can still derive the

[1] Recall that priority was modelled in the Requirements Ontology as classes (ro:LowPriority, ro:MediumPriority . . .) therefore, we are mapping rdf:type.

ordinal scale of the priority values from the value mapping (which uses rvl:sourceValueOrderedSet to define a set of ordered priority values). In order to calculate the scale of measurement from the value mapping, we had to extend the effectiveness constraint from Listing 8.3 following the decision diagram from Sect. 7.4.1.
Next, we add a further mapping from ro:hasResponseTimeInMs to *labelling*, i. e., whenever information on the response time is available, a label is attached to the graphic object representing the requirement.

```
:IconLabelMappingForHasResponseTime
  a rvl:PropertyMapping ;
  rvl:sourceProperty ro:hasResponseTime ;
  rvl:targetObjToObjRelation vg:Labeling_Relation .
```

This label is an iconic label and has in turn a textual label stating the value of the response time. Labelling of labels can be achieved by combining multiple mappings using the submapping mechanism of RVL. The three labelling steps are shown below:



First, resources are assigned a graphic object playing the role of a label. This object has a default shape (here a square). By applying two submappings to the label, we refine its appearance. With the first submapping, an rvl:ResourceMapping, we change its shape to the clock symbol.

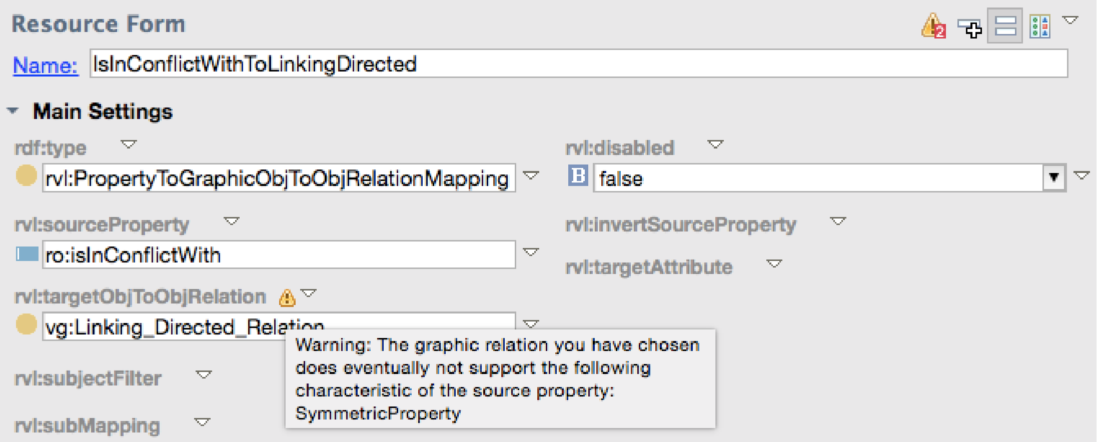
```
:IconLabelMappingForHasResponseTime
  a rvl:PropertyMapping ;
  rvl:sourceProperty ro:hasResponseTime ;
  rvl:targetObjToObjRelation vg:Labeling_Relation ;
  rvl:subMapping
    [ rvl:subMapping-mapping :IconShapeMapping ;
      rvl:subMapping-onRole vg:labelling_label ;
      rvl:subMapping-onTriplePart rdf:predicate ],
    [ rvl:subMapping-mapping :TextLabelingOfLabel ;
      rvl:subMapping-onRole vg:labelling_label ;
      rvl:subMapping-onTriplePart rdf:subject ] .

:IconShapeMapping
  a rvl:ResourceMapping ;
  rvl:sourceValue ro:hasResponseTime ;
  rvl:targetValue common-shapes:Clock .
```

The second submapping attaches another labelling relation to produce the textual label (for the concrete time value).

```

:TextLabelingOfLabel
  a rvl:PropertyMapping ;
  rvl:sourceProperty ro:hasResponseTime ;
  rvl:targetObjToObjRelation vg:Labeling_Relation ;
  rvl:subMapping [
    rvl:subMapping-mapping :LabelingTextIdentityMapping ;
    rvl:subMapping-onRole vg:labelling_label ;
    rvl:subMapping-onTriplePart rdf:subject ] .

:LabelingTextIdentityMapping
```

```
  a rvl:IdentityMapping ;
  rvl:sourceProperty ro:hasResponseTime ;
  rvl:targetAttribute vg:text_value .
```

Now, we need to add a further mapping to encode ro:isInConflictWith. If we try to add a mapping from this property to *directed linking*, this will issue another warning, since the system evaluates the property characteristics defined with OWL, stating that ro:isInConflictWith is an owl:SymmetricProperty. Based on a SPIN constraint defining that the symmetry of properties cannot be expressed by *directed linking*, the system suggests to choose *undirected linking* instead. The corresponding warning and quickfix as suggested by prototype P1 is given below:



Resource Form

Name: IsInConflictWithToLinkingDirected

Main Settings

rdf:type: rvl:PropertyToGraphicObjToObjRelationMapping

rvl:disabled: false

rvl:sourceProperty: ro:isInConflictWith

rvl:invertSourceProperty

rvl:targetAttribute

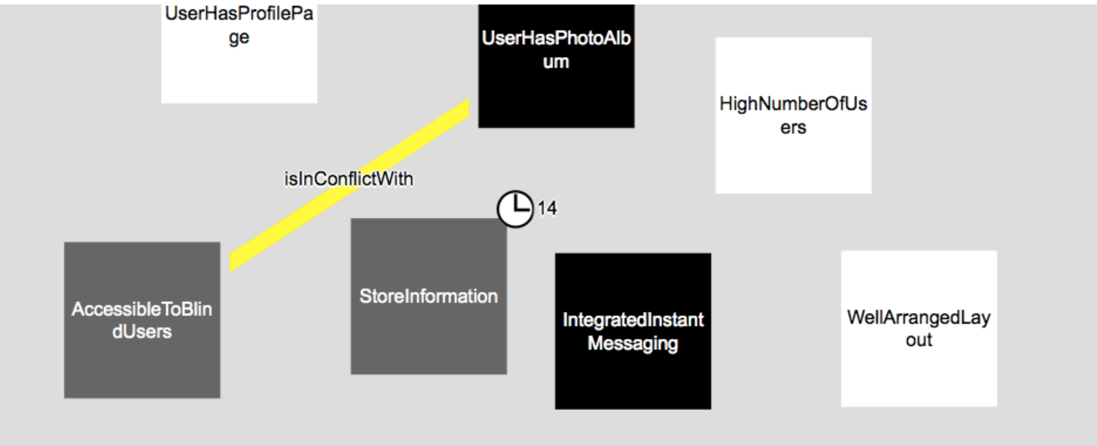
rvl:targetObjToObjRelation: vg:Linking_Directed_Relation

Warning: The graphic relation you have chosen does eventually not support the following characteristic of the source property: SymmetricProperty

rvl:subjectFilter

rvl:subMapping

Adding a mapping to undirected linking results in the graphic below. Unlike in the introductory example (Fig. 1.3), the graphic relation *undirected linking* is not rendered as a double-headed arrow, but as a simple line without arrow-heads, since we do not yet offer a mechanism to select from multiple options for rendering the same graphic relation.



For completeness, we also give the brief listing of this mapping:

```
:IsInConflictWithMapping
  a rvl:PropertyMapping ;
  rvl:sourceProperty ro:isInConflictWith ;
  rvl:targetObjToObjRelation vg:Linking_Undirected_Relation .
```

Finally, in order to completely recreate the graphic from the introductory example, we would need to add a last mapping from ro:isRefinentOf to *containment*. While this is supported by RVL and can be represented as an AVM (cf. Fig. 6.6), we do not yet provide a corresponding D3 rendering. Things to consider when implementing such a rendering include attachment-points for connectors and repositioning container labels.

Chapter 10

Conclusions

In this last chapter, we summarise the contributions of this thesis and the results of the constructive evaluation. Finally, we review the research questions, conclude to what extent they can be answered by our results, and point to future work.

10.1 Contributions

In this thesis, we make the following main contributions:

C-1 The OGVIC approach to **Ontology-Driven, Guided Visualisation Supporting Explicit and Composable Mappings**

C-2 A formalisation of graphic terms and knowledge in the **Visualisation Ontology (VISO)** [PV13, VP11]

C-3 The principle of the platform-independent **Abstract Visual Model (AVM)**, supporting a fine-grained, role-based description of the composed graphics

C-4 The declarative **RDFS/OWL Visualisation Language (RVL)** [Pol13]

C-5 A **detailed analysis of the state of the art** in the field of visualisation approaches and languages used for visualisation and RDF-presentation

C-6 Three **prototypical implementations**, which cover the essential parts of the OGVIC approach and show its feasibility [Pol15]

The first contribution of this thesis *(C-1)* is the description of OGVIC, an approach to ontology-driven visualisation that not only allows for visualising ontologies, but also uses ontologies to guide users when visualising ontological data (Sect. 8.1). We realise guidance by evaluating constraints defined by the mapping language (RVL) as well as an external fact base, the VISO module *VISO/facts/empiric*. Based on the constraints and the formalised visualisation knowledge, warnings are issued and mapping options are suggested. That means, we discourage the construction of mappings that are »non-optimal« according to perceptual rankings formalised in the fact base and suggest better, e. g., more effective mappings instead. The *VISO/facts/empiric* module is part of our second contribution, the VISO ontology:

With the VISO ontology *(C-2)*, a comprehensive, collaborative collection of visualisation knowledge has been created (Chapter 5). Work on this ontology included a detailed survey of existing ontologies, classifications and other models in the field of visualisation and graphics.

Besides for the OGVIC approach, VISO is also the foundation for an approach to recommend visualisation components [VPGM12] implemented as part of VizBoard [VPM13].

The Abstract Visual Model (AVM; Chapter 6) is the third contribution of this thesis *(C-3)*. It is novel, since it is the first approach to formally represent and »*synthesise*« a graphic following a role-based approach, inspired by the one used by Engelhardt for the *analysis* of graphics. The justification for the AVM – which, at first glance, may appear as an unnecessary indirection – is threefold: First, in the terminology of the model-driven paradigm, it plays the role of a platform-independent model and thereby supports platform variability. Second, it enables introspection on the level of graphic relations and supports tracing back to the represented data, unlike, for example, SVG does. This paves the ground for guidance as we suggest it with the *OGVIC* approach. Third, the role-based composition of graphics in the AVM allows for defining the submapping mechanism for the composition of visual mappings in RVL.

The declarative RDFS/OWL Visualisation Language (RVL; Chapter 7) is the fourth contribution *(C-4)*. Being specific to RDFS and OWL concepts, it is focused on the visualisation of knowledge specified in these languages. An important aspect of RVL is that the mappings from the relations in the data to the graphic relations are made explicit. Further, we ease reuse and sharing of visual mappings by two means; the composability of the mappings and the fact that RVL mappings are RDF data themselves. Hence they can be conveniently stored, published, documented and dereferenced just like the data they are meant to visualise.

To distinguish OGVIC and RVL from existing approaches and languages, a detailed analysis of the state of the art in visualisation systems, visualisation languages and RDF presentation languages was done (Chapter 4), which can be seen as an additional contribution *(C-5)*. Comparing ten approaches by 29 criteria, we showed that the uniqueness of the approach emerges from the fact that we combine ontology-driven guidance with a concept for composable visual mappings.

As a last contribution *(C-6)*, we presented three prototypical implementations[1]. The prototypes have been built with technologies within the RDF technical space (Sect. 8.3, 8.4, and 8.5). The development of three prototypes was due to technical reasons and the platform diversity is a side-effect. However, the experiments also confirm that we achieved some degree of platform variability – artefacts, like the RVL mappings or the AVM, can be exchanged between all three prototypes, which allows for combining their strengths. While none of the prototypes implements the complete architecture proposed by the OGVIC approach, each process step is covered by at least one of the prototypes (except for advanced filtering and selection). The prototypes do not implement a wizard-like step-by-step guidance process as we suggested with the introductory example. However, adding such behavior to P3 is possible based on existing models – the AVM, the mapping model and the VISO ontology. Only the option to choose between multiple renderings of the same graphic relation would require to extend the current modelling, since currently we assume that this choice is left to the system. Prototype P3 includes a Java implementation of the RVL language showing that it is automatable.

10.2 Constructive Evaluation

In Chapter 9, we documented the visualisation cases and requirements that are covered by the current specifications and prototypes. We also documented the results of a constructive evaluation of both the RVL/VISO/AVM specifications and prototype P3 against the sketches that we did as part of our case studies. We see the constructive evaluation as a first step to ensure that our approach meets our requirements and is able to cover a large number of the visualisation cases and sketches that we did. In summary, although many visualisation cases are not yet completely covered and no prototype meets all our requirements, we can already generate a large portion of the sketches from our case studies. Two thirds of the 37 sketches can

[1] One of the prototypes was developed as student work, supervised by the author, cf. Sect. 8.4.

be almost completely or completely specified; half of the sketches can be almost implemented or completely implemented with prototype P3.

Although OGVIC is intended to be domain-agnostic, only a user study in various domains could proof the general usefulness of the approach. However, the fact that we derived our requirements from use cases of various domains, suggests that OGVIC is applicable to other domains as well.

10.3 Research Questions

Let us recall the research questions formulated in Sect. 3.2. Research question Q-1 asked for a way to »define composable and shareable mappings from ontological data to visual means«. This has been answered to a large extent: RVL can be used to define composable, shareable mappings. The high degree of composability we aim at, and already achieve up to the level of the AVM model, is limited, though, by how the prototypes render the AVM, since they use fix graphic types in their current state (see future work on the prototypes described in Sect. 8.6.6). The requested target model (Q-1.2), is provided by the VISO/graphic ontology. A further subquestion (Q-1.1) was: »Which ontology constructs do we want to map and onto which visual means?« This has been answered for the scope of the example ontologies from our case studies (Chapter 3) and deserves a further requirement analysis for additional domains. However, with the rich palette of graphic relations formalised in VISO/graphic, a good foundation has been laid for covering additional visual means on demand.

Research question Q-2 asked for a way to »guide the visual mapping of ontological data« and has mostly been answered. The guidance process is flexible, since it is based on single exchangeable rules. With our prototypes, we showed how the composition of visual mappings can be supported by tools that display warnings and error messages for non-optimal mappings. However, none of the prototypes reaches the guidance level of »recommendations«. One subquestion (Q-2.2) was on how we can »formalise expert visualisation knowledge«. In the OGVIC approach, visualisation knowledge is formalised with the VISO/facts ontology and the corresponding example knowledge base VISO/facts/empiric. The other subquestion (Q-2.1) asked whether visualisation could »benefit from the rich semantics of ontological data« and whether the rich semantics of ontological data could »help to allow for other visual paradigms than the node-link paradigm«. While we describe a guidance mechanism based on rules, constraints and VISO-based facts, we cannot say with certainty, whether the semantics of ontological data facilitate the use of visual paradigms beyond node-link diagrams. Nevertheless, we showed that it is possible to benefit from relation characteristics such as »symmetry« (cf. Sect. 9.4). In Chapter 3 we listed more characteristics of ontological data that can be used to derive visual mapping suggestions.

With respect to the main question of how to support the tailor-made, effective and reusable visualisation of ontologies, the OGVIC approach offers the option to synthesise tailor-made graphics based on reusable components. In the previous chapter, we demonstrated how graphics can be varied, if complex mappings are composed from simple ones. Also the effectiveness of the employed visual means is considered for guiding the user. The efficiency of the OGVIC approach, including the suggested guidance process, needs to be measured in future user studies.

10.4 Transfer to Other Models and Constraint Languages

While we picked RDF-based ontological data as our concrete data model, the general principles described in this thesis are, to a large extent, applicable to **other data models, such as Ecore-based models** as they are common in the software modelling community. Similarly, the formally stored visualisation knowledge in VISO – and partly also the RVL mappings – may be used in non-Semantic-Web contexts, if adapters are built that relate existing systems to the concepts defined in VISO and RVL. For example, RVL mappings could be used as an exchange format by non-Semantic-Web visualisation design systems. It could also be considered to extend or generalise RVL to define visual mappings for **other graph models than the RDF model, for example, property graphs** as they are used in databases like Neo4j[2] [Mil13]. Also Wikidata [Vra12], which could become an important source of knowledge, uses graphs that offer temporal context. Alternatively, RVL can always be applied to an RDF export that represents a »snapshot« of these richer graphs. Using property graphs could even simplify some aspects of OGVIC, for example n-ary, weighted graphic relations in the AVM could more elegantly be represented by a property graph – at the cost of a technological break. In this context, the hypergraph approach of Minas [Min00] should also be taken into account (cf. Sect. 6.11).

In our approach, guidance is realised by evaluating constraints that are currently defined by SPIN constraints. This has the benefit that we can easily refer to both domain concepts and graphic concepts and infer new visualisation knowledge in the ontology technological space. Even if SPIN does not succeed on the long run and is replaced by other formalisms, the constraints that we defined may easily be reused, since they are internally based on the well-established W3C standard SPARQL. Our decision to use a mechanism like SPIN, instead of OWL, is supported by the fact that currently two groups, the W3C *RDF Data Shapes Working Group*[3] and the Dublin Core »RDF Application Profiles« Task Group[4] (focus on requirements of such a language) work on the standardisation of technologies that are similar to SPIN. A SPARQL-based language called *Shapes Constraint Language* (SHACL) is discussed by the RDF Data Shapes Working Group, which has many similarities with SPIN [SHA15, SHA16].

In the following, we briefly note what the OGVIC approach does *not* cover. We differentiate between issues that are generally out of scope and those that have not been in the focus of this thesis but could be done as future work.

10.5 Limitations

No focus on tabular and statistical data. We do not focus on homogeneous, tabular data and common graphic types such as bar and pie charts, which are often used for this kind of homogeneous data. Also, we do not cover the integration of statistic functions as they are often applied on tabular data such as measurements. As we have shown in Chapter 4, there are already many approaches for these scenarios.

No visual language. We do not define a graphic syntax for RVL itself. That means, we do not use advanced visualisations to define the mappings (unlike a graphical query language such as RDF-GL [HMFK10]).

10.6 Future Work

Extension and combination of prototypes. The prototypes need to be combined and should be extended to allow for real world scenarios and user-studies. This includes the extension to further process steps such as **data selection and filtering** and the implementation of **better tools supporting higher levels of guidance** (Sect. 8.1.2). Furthermore, a **hybrid implementation of an RVL interpreter** should be considered, which uses graph transformations for some of the necessary transformations. Details on future work concerning the implementation of a comprehensive prototype were given in Sect. 8.6.

[2] Neo4j. http://neo4j.com, accessed: 12.12.2015.

[3] RDF Data Shapes Working Group. http://www.w3.org/2014/data-shapes/wiki/Main_Page, accessed: 10.06.2015.

[4] RDF Application Profiles Task Group at the Dublin Core Metadata Initiative. http://wiki.dublincore.org/index.php/RDF_Application_Profiles/, accessed: 10.06.2015.

Advanced, topology-aware composition. We support a basic composition of mappings. Still, there are composition cases (Sect. 8.2.1) that cannot be defined with RVL or will not be considered for guidance. To decide, which graphic relations may be used in combination, the system could further benefit from an advanced analysis of the instance data's topology (cf. Sect. 5.7.3). Characteristics such as planarity or the occurrence of cycles should be offered by an additional graph analysis module. Here, collaboration with graph topology experts could be beneficial. With the AVM, we offer a model that is rich enough to support running such an analysis also for the generated graphic. The topological results can then be used to decide on the applicability of graphic relations that put specific constraints on the data, such as *building*, or the composition of *containment* and *linking*.

Fresnel and CSS integration. The Fresnel vocabulary for structuring document-like parts of the graphics should be integrated with the OGVIC approach, particularly with the RVL language. A basic CSS support was implemented (roles of graphic objects are provided as hooks), but the use of CSS for styling texts and graphics needs to be further integrated with RVL. For example, all named colours in RVL should be replaced by their CSS equivalents.

Reduced interaction times. It has to be further investigated, how the model-driven architecture of OGVIC can be modified to further support interaction with quick responses of the GUI. Users will benefit from instantly seeing changes in the graphic as a response to their changes to the configuration. This has been described in the context of visual analytics by Kerren and Schreiber [KS12]. Using iterative and incremental transformations could be one concept towards shorter interaction times for complex graphics. Transforming OGVIC towards a full MVC architecture has been discussed as future work in Sect. 8.6.5.

Guided configuration of rendering. At the moment, the rendering of the graphics cannot be configured by end-users. As rendering is just another step in the visualisation process, also the rendering could be configured at runtime. This could be used to switch between different target platforms (e. g., switch from 2D SVG-rendering to X3D-rendering). Also for this process step, guidance could be provided to help mapping abstract graphic relations and attribute values to concrete parameters of the output platform. Guidance for this process step is also required to implement the last system interaction from our example dialogue that we used to motivate a guided visualisation system in the introduction.

Editing source data. The editing of the source data is another case of interaction in the visualisation process, which we excluded from the requirements *(L-1)* for the scope of this thesis. Further investigation will be necessary to enable the »back-transformation« of changes to the graphic representation back to the data that it represents. However, the fact that we aim at unambiguous visual mappings – each mapping should be bidirectional in the sense that it must be clearly decodeable by human perception – may be a good basis for editing. It has to be investigated, whether experiences and ideas from round-trip engineering [Aß03a], e. g., [Sei11], can be applied to visualisations and whether the mapping definitions need to be prepared in order to allow for bidirectional transformations. Round-trip engineering has already been applied in the context of model-driven development of 3D web-applications by Lenk et al. [LVJ12]. Also our general architecture promotes editing, e. g., trace links can easily be created to the underlying data for each graphic element. Among the examined related work, only Protovis by Bostock and Heer (Sect. 4.2.1) and the RDF Editing Metamodel (REMM; Sect. 4.3.1) by Rauschmeyer offer a mapping description that explicitly supports editing.

Support the sharing of mappings. To foster the use of RVL for describing visualisation mappings, a platform for sharing and reusing recommended visualisation settings in RVL could be built by a community project.

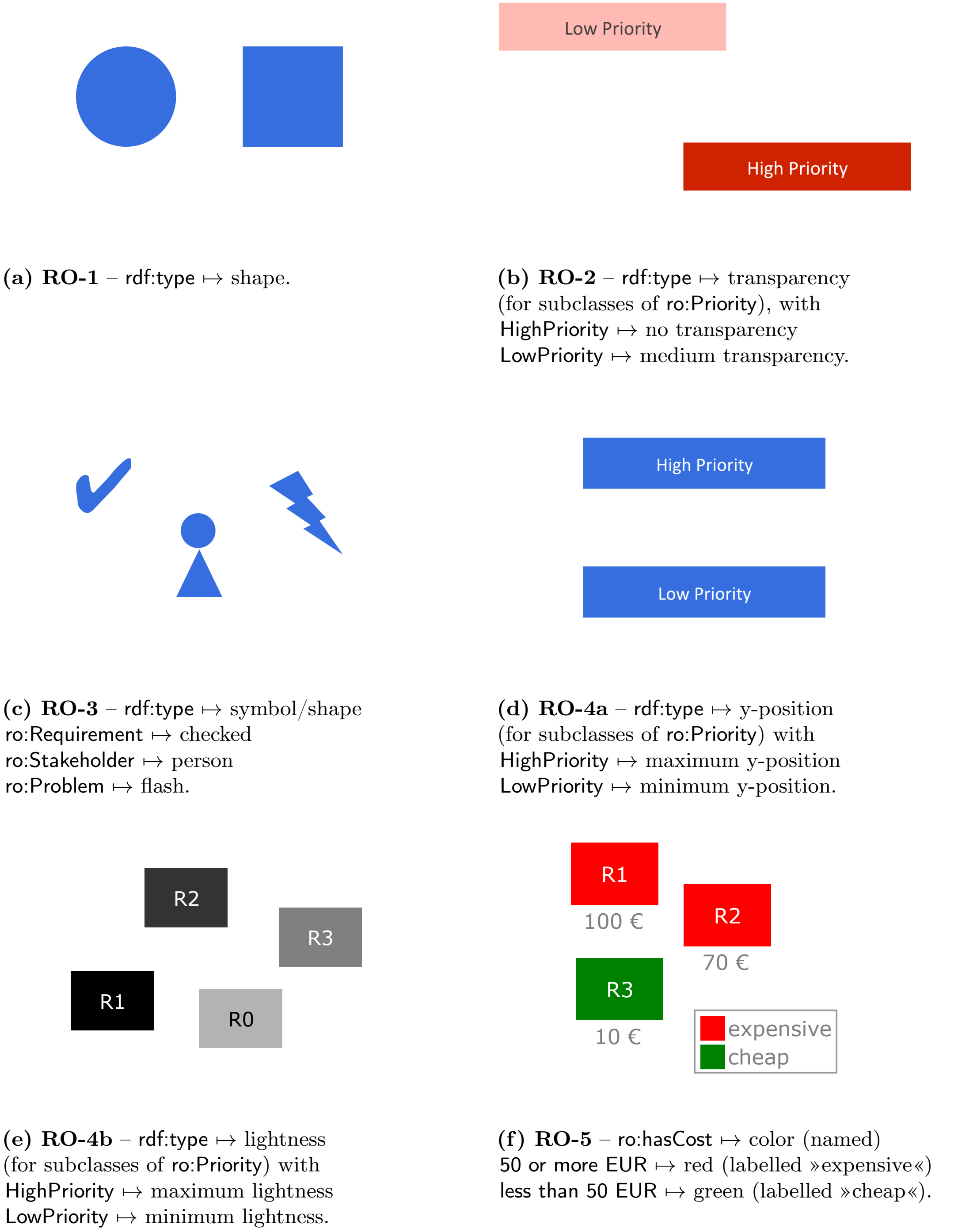
Extension to other senses. We did not take other senses such as audio or tactile into consideration, and limited ourselves to the visual sense. However, graphic has a broader meaning as Wilkinson [Wil05] states: »*there is nothing in the definition of a graphic* [..] *to limit it to vision*« and »*touch, hearing, and other senses can be used to convey information with as much as detail and sensitivity as can vision*«. Therefore, to allow for future extensions to other senses, we use the term »graphic« instead of »visual« wherever possible (e. g., this is the reason, we use »graphic attribute« instead of »visual attribute«).

Appendices

Appendix A

Case Study Sketches

The following sketches have been done in order to identify mapping cases that occur, when data from the case studies' ontologies is visualised using graphic representation types that are common in the respective domains.



(a) RO-1 – rdf:type ↦ shape.

(b) RO-2 – rdf:type ↦ transparency (for subclasses of ro:Priority), with
HighPriority ↦ no transparency
LowPriority ↦ medium transparency.

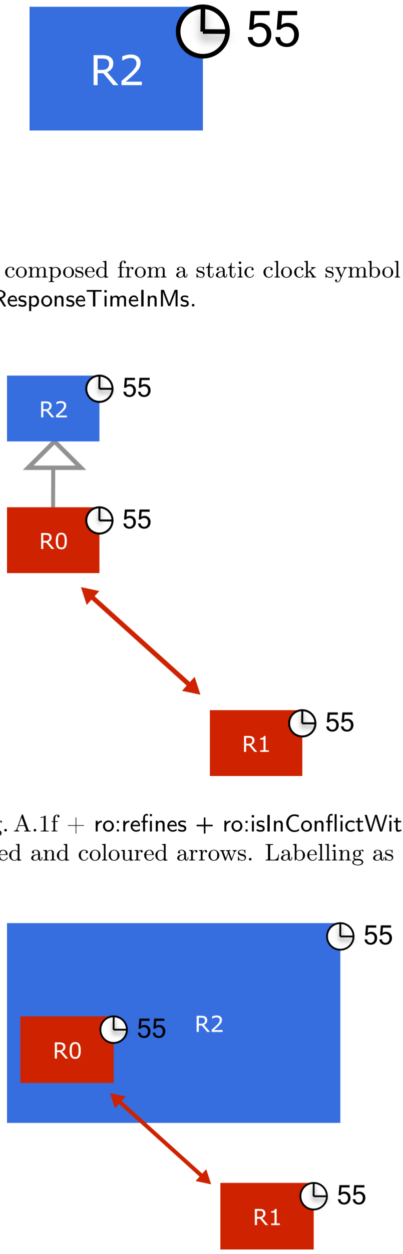
(c) RO-3 – rdf:type ↦ symbol/shape
ro:Requirement ↦ checked
ro:Stakeholder ↦ person
ro:Problem ↦ flash.

(d) RO-4a – rdf:type ↦ y-position (for subclasses of ro:Priority) with
HighPriority ↦ maximum y-position
LowPriority ↦ minimum y-position.

(e) RO-4b – rdf:type ↦ lightness (for subclasses of ro:Priority) with
HighPriority ↦ maximum lightness
LowPriority ↦ minimum lightness.

(f) RO-5 – ro:hasCost ↦ color (named)
50 or more EUR ↦ red (labelled »expensive«)
less than 50 EUR ↦ green (labelled »cheap«).

Figure A.1: Sketches for the case studies' ontologies – Domain of software technology I.

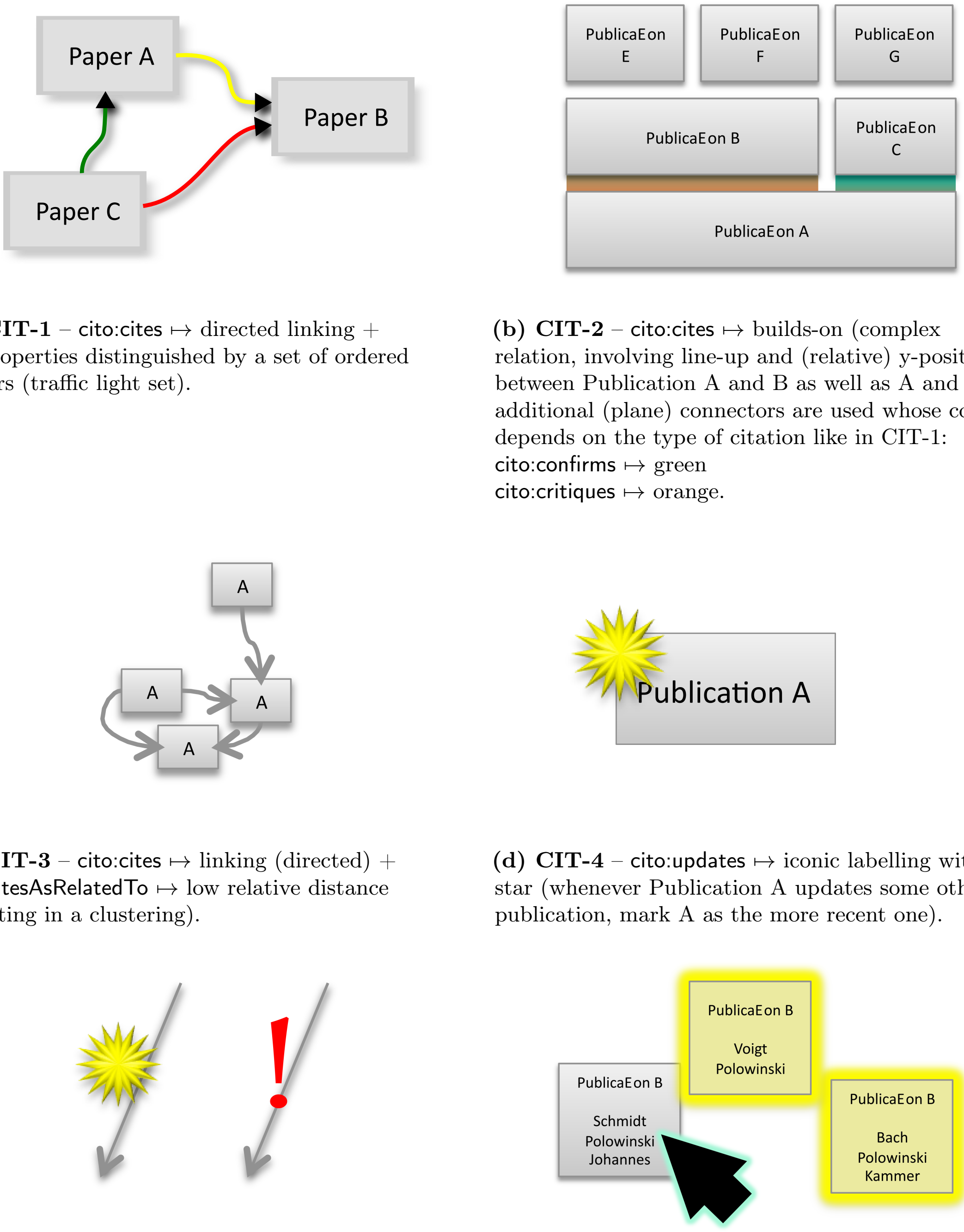


(a) RO-8 – A label, composed from a static clock symbol and a text value encodes ro:hasResponseTimeInMs.

(b) RO-6 – Like Fig. A.1f + ro:refines + ro:isInConflictWith ↦ linking with differently shaped and coloured arrows. Labelling as in (RO-8).

(c) RO-7 – Like Fig. A.2b, but ro:refines is now mapped to containment instead of linking.

Figure A.2: Sketches for the case studies' ontologies – Domain of software technology II.



(a) CIT-1 – cito:cites ↦ directed linking + subproperties distinguished by a set of ordered colours (traffic light set).

(b) CIT-2 – cito:cites ↦ builds-on (complex relation, involving line-up and (relative) y-position; between Publication A and B as well as A and C additional (plane) connectors are used whose color depends on the type of citation like in CIT-1:
cito:confirms ↦ green
cito:critiques ↦ orange.

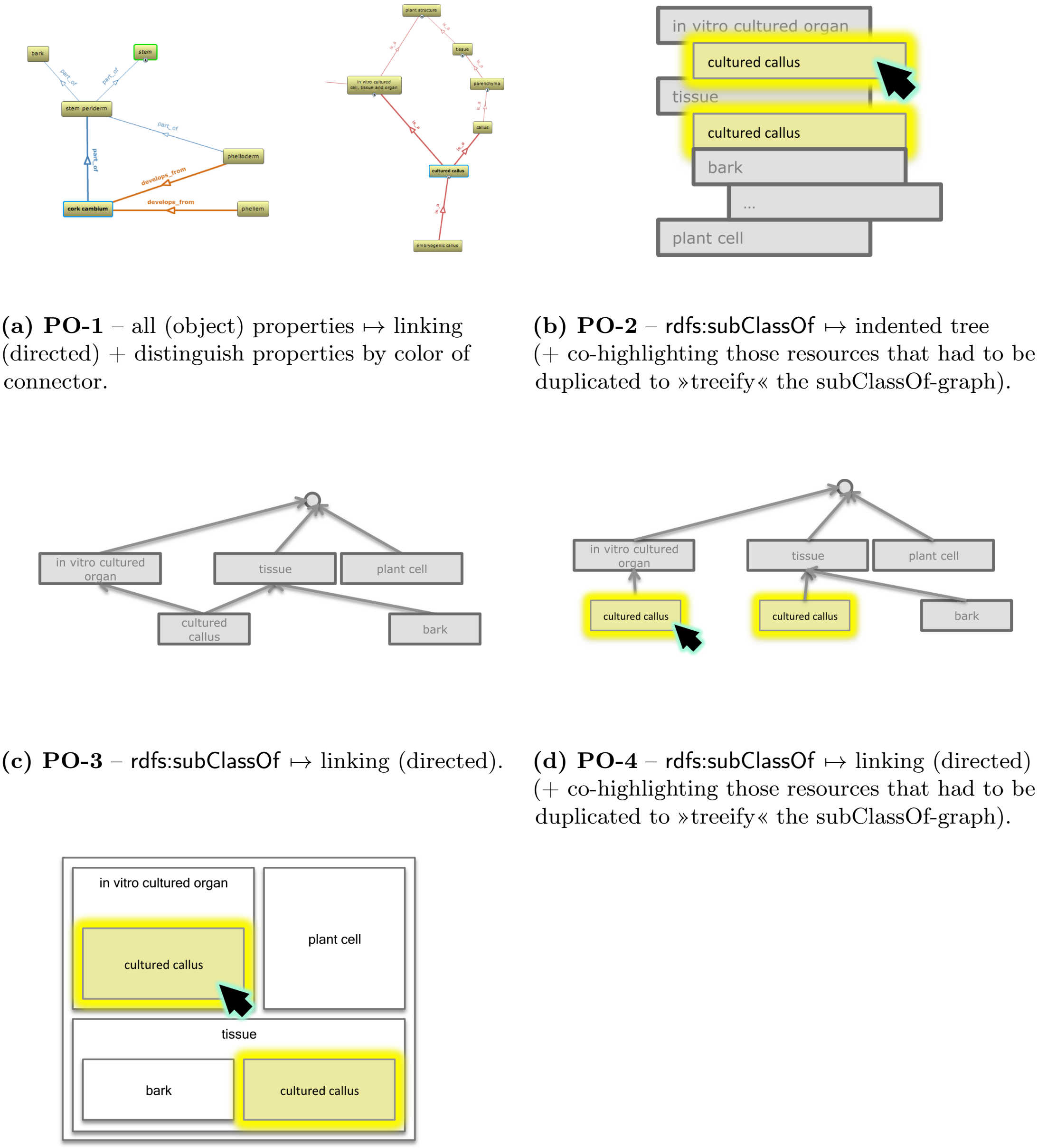
(c) CIT-3 – cito:cites ↦ linking (directed) + cito:citesAsRelatedTo ↦ low relative distance (resulting in a clustering).

(d) CIT-4 – cito:updates ↦ iconic labelling with a star (whenever Publication A updates some other publication, mark A as the more recent one).

(e) CIT-5 – cito:cites ↦ directed linking + iconic labels on the connector to distinguish subproperties of cites:
cito:updates ↦ star
cito:disagreedBy ↦ note of exclamation.

(f) CIT-6 – cito:sharesAuthorWith ↦ co-highlight publications having at least one author in common with the selected publication are highlighted.

Figure A.3: Sketches for the case studies' ontologies – Domain of publishing.



(a) PO-1 – all (object) properties ↦ linking (directed) + distinguish properties by color of connector.

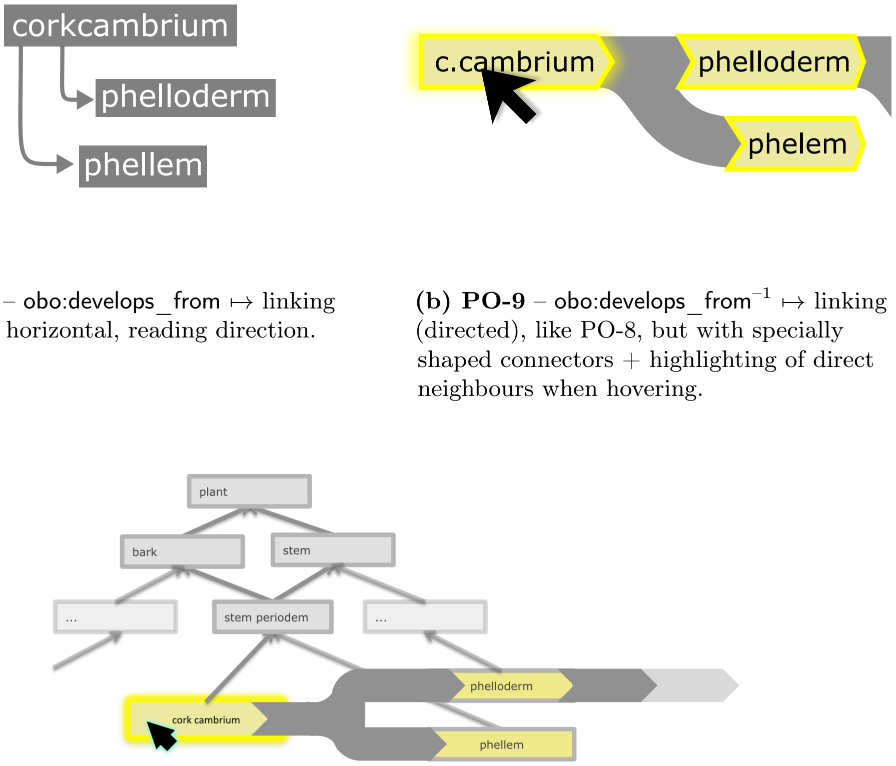
(b) PO-2 – rdfs:subClassOf ↦ indented tree (+ co-highlighting those resources that had to be duplicated to »treeify« the subClassOf-graph).

(c) PO-3 – rdfs:subClassOf ↦ linking (directed).

(d) PO-4 – rdfs:subClassOf ↦ linking (directed) (+ co-highlighting those resources that had to be duplicated to »treeify« the subClassOf-graph).

(e) PO-5 – rdfs:subClassOf ↦ containment (+ co-highlighting those resources that had to be duplicated to »treeify« the subClassOf-graph).

Figure A.4: Sketches for the case studies' ontologies – Domain of biology (Plant Ontology) II. PO-6 and PO-7 are not shown, since they use the same graphics as PO-2 resp. PO-3 but for the po:part_of relation.

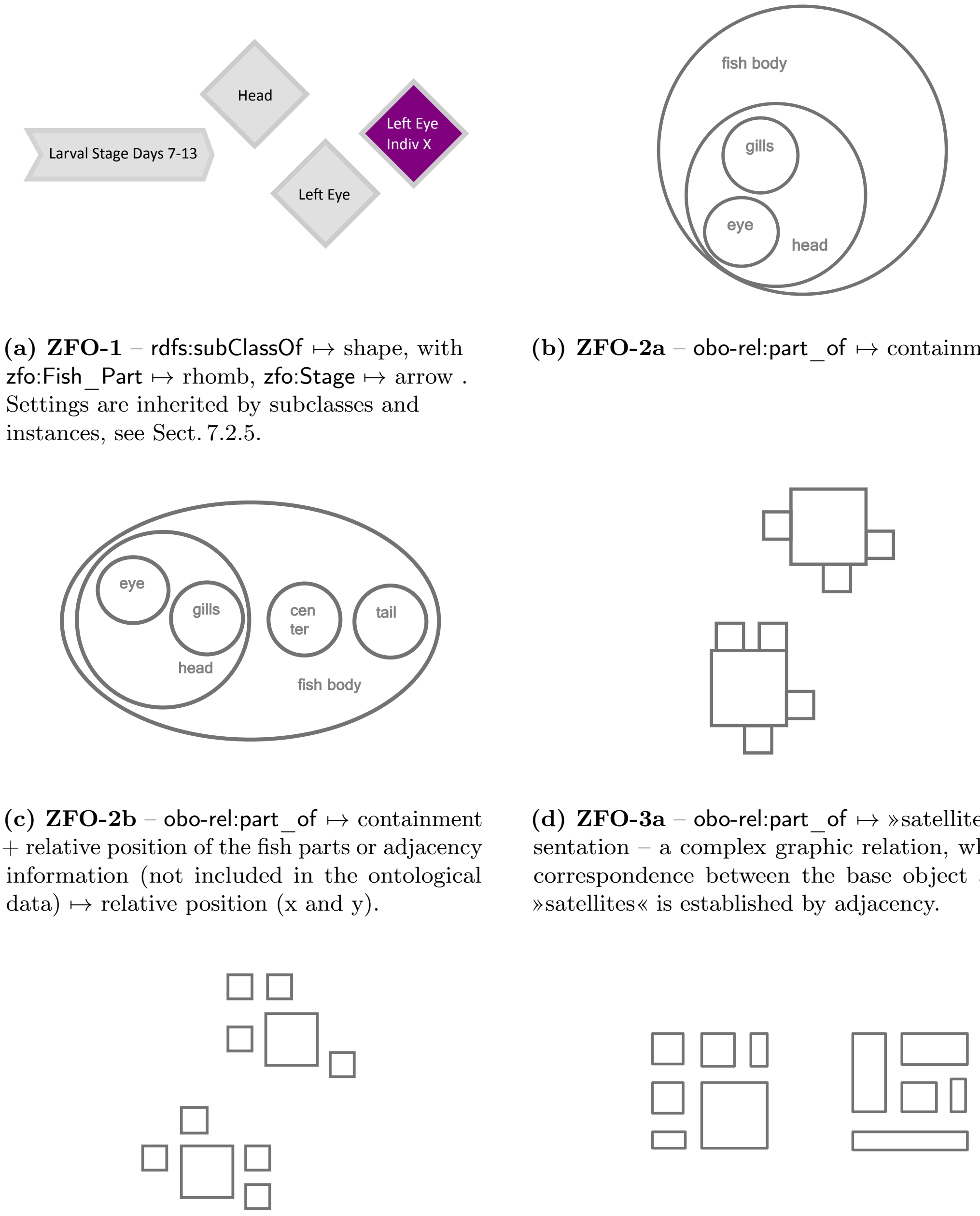


(a) PO-8 – obo:develops_from ↦ linking (directed), horizontal, reading direction.

(b) PO-9 – obo:develops_from^{-1} ↦ linking (directed), like PO-8, but with specially shaped connectors + highlighting of direct neighbours when hovering.

(c) PO-10 – Attempt to interactively connect two linking-structures, one representing the obo:develops_from relation (PO-9) and the other representing a hierarchy of plant parts (PO-7; not shown).

Figure A.5: Sketches for the case studies' ontologies – Domain of biology (Plant Ontology) II. PO-6 and PO-7 are not shown, since they use the same graphics as PO-2 resp. PO-3 but for the po:part_of relation.



(a) ZFO-1 – rdfs:subClassOf ↦ shape, with zfo:Fish_Part ↦ rhomb, zfo:Stage ↦ arrow . Settings are inherited by subclasses and instances, see Sect. 7.2.5.

(b) ZFO-2a – obo-rel:part_of ↦ containment.

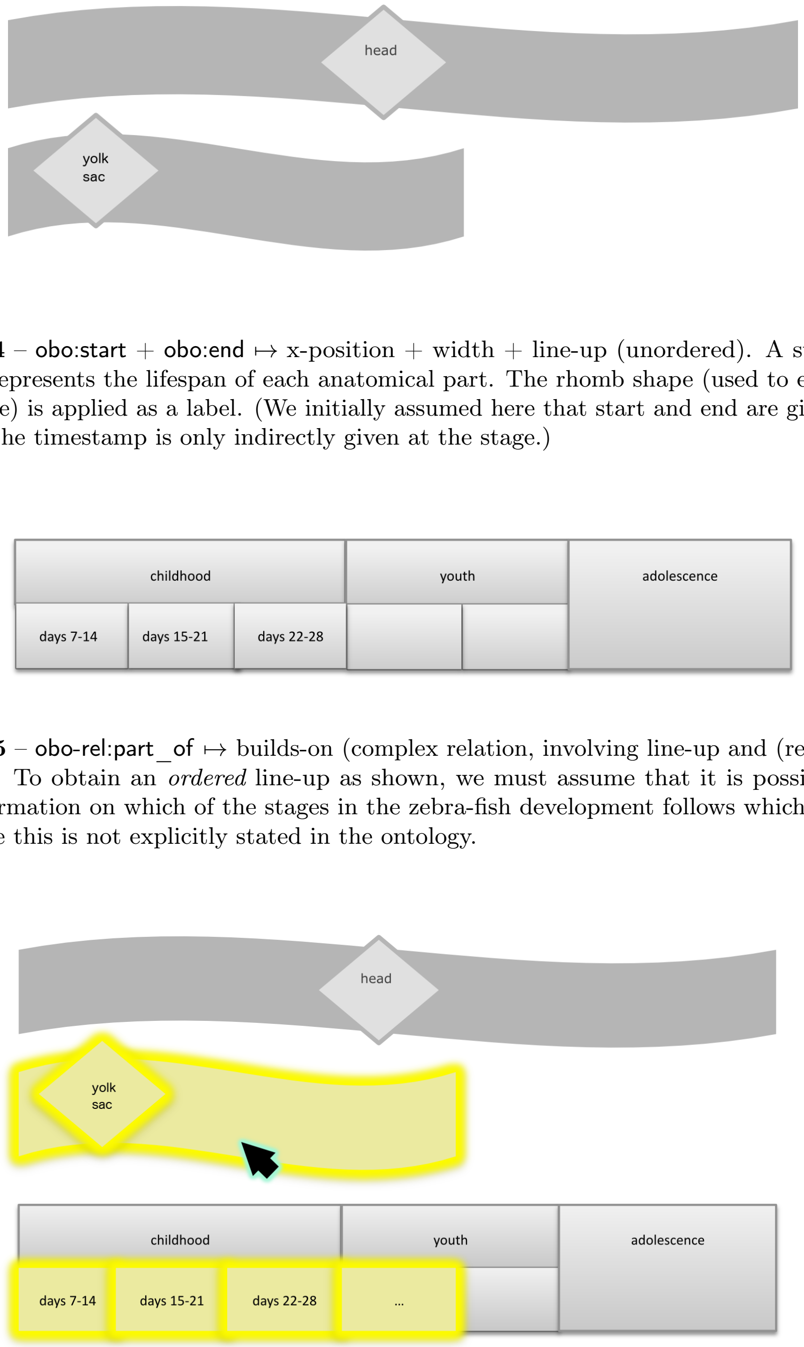
(c) ZFO-2b – obo-rel:part_of ↦ containment + relative position of the fish parts or adjacency information (not included in the ontological data) ↦ relative position (x and y).

(d) ZFO-3a – obo-rel:part_of ↦ »satellite« representation – a complex graphic relation, where the correspondence between the base object and the »satellites« is established by adjacency.

(e) ZFO-3a1 – Same as ZFO-3a, but the correspondence between the base object and the »satellites« is established by a low relative distance.

(f) ZFO-3b – Part-whole relation in the Zebrafish's anatomy visualised by a shape emerging from multiple clustered graphic objects. (Similar to a tree-map, but area does not encode information. Additionally, (here) limited to a depth of one obo-rel:part_of relation.)

Figure A.6: Sketches for the case studies' ontologies – Domain of biology (Zebra Fish Anatomy Ontology) I.

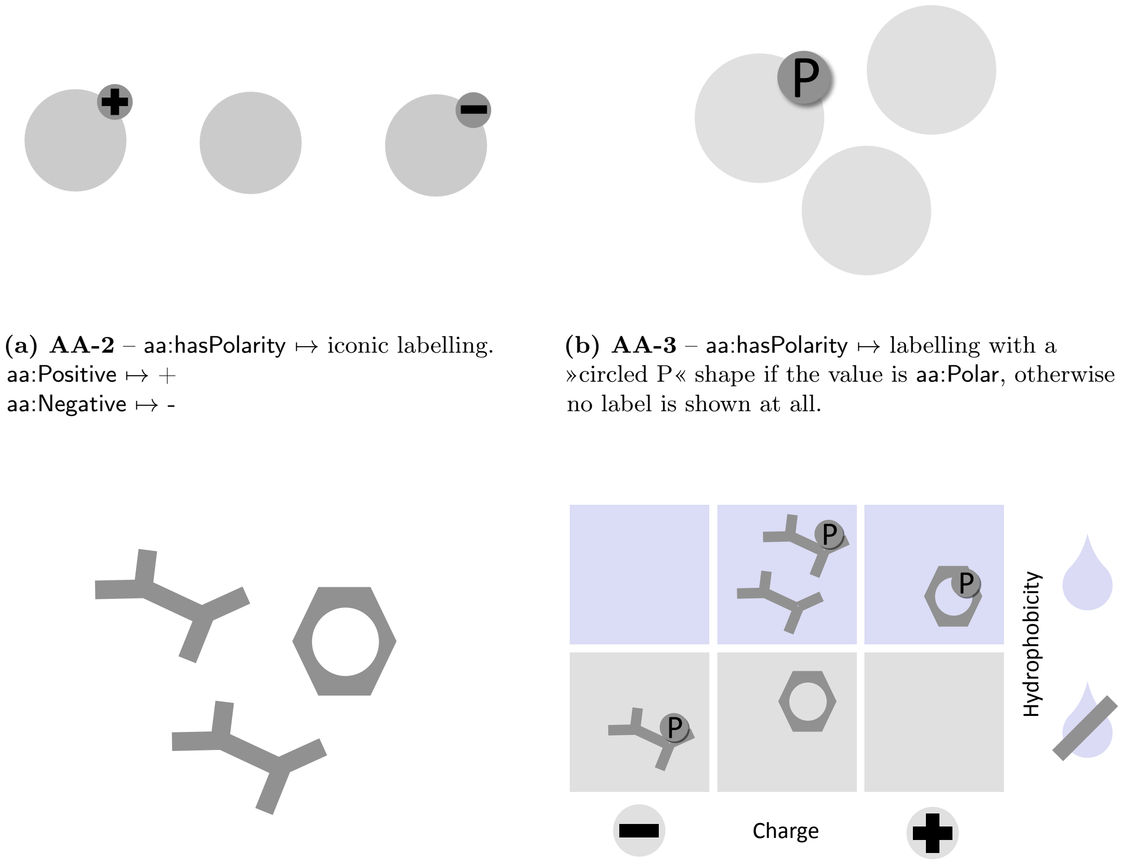


(a) **ZFO-4** – obo:start + obo:end ↦ x-position + width + line-up (unordered). A swifted rectangle represents the lifespan of each anatomical part. The rhomb shape (used to encode parts before) is applied as a label. (We initially assumed here that start and end are given in days, but the timestamp is only indirectly given at the stage.)

(b) **ZFO-5** – obo-rel:part_of ↦ builds-on (complex relation, involving line-up and (relative) y-position. To obtain an *ordered* line-up as shown, we must assume that it is possible to derive information on which of the stages in the zebra-fish development follows which other stage, since this is not explicitly stated in the ontology.

(c) **ZFO-6** – Combination of ZFO-4 and ZFO-5 in a multipanel display. On selecting one of the time spans in the upper graphic, the corresponding start to end development stages in the hierarchy of stages are highlighted.

Figure A.7: Sketches for the case studies' ontologies – Domain of biology (Zebra Fish Anatomy Ontology) II.



(a) AA-2 – aa:hasPolarity ↦ iconic labelling.
aa:Positive ↦ +
aa:Negative ↦ -

(b) AA-3 – aa:hasPolarity ↦ labelling with a »circled P« shape if the value is aa:Polar, otherwise no label is shown at all.

(c) AA-4 – aa:hasSideChainStructure ↦ shape.

(d) AA-5 – Tabular representation of amino acids using two spatial dimensions to segregate amino acids by aa:hasCharge and aa:hasHydrophobicity. Additionally, mappings from aa:hasSideChainStructure and aa:hasPolarity to shape (Fig. 3.2b) and labelling (Fig. 3.3d) are reused.

Figure A.8: Sketches for the case studies' ontologies – Domain of biology (Amino Acids Ontology).

Appendix B

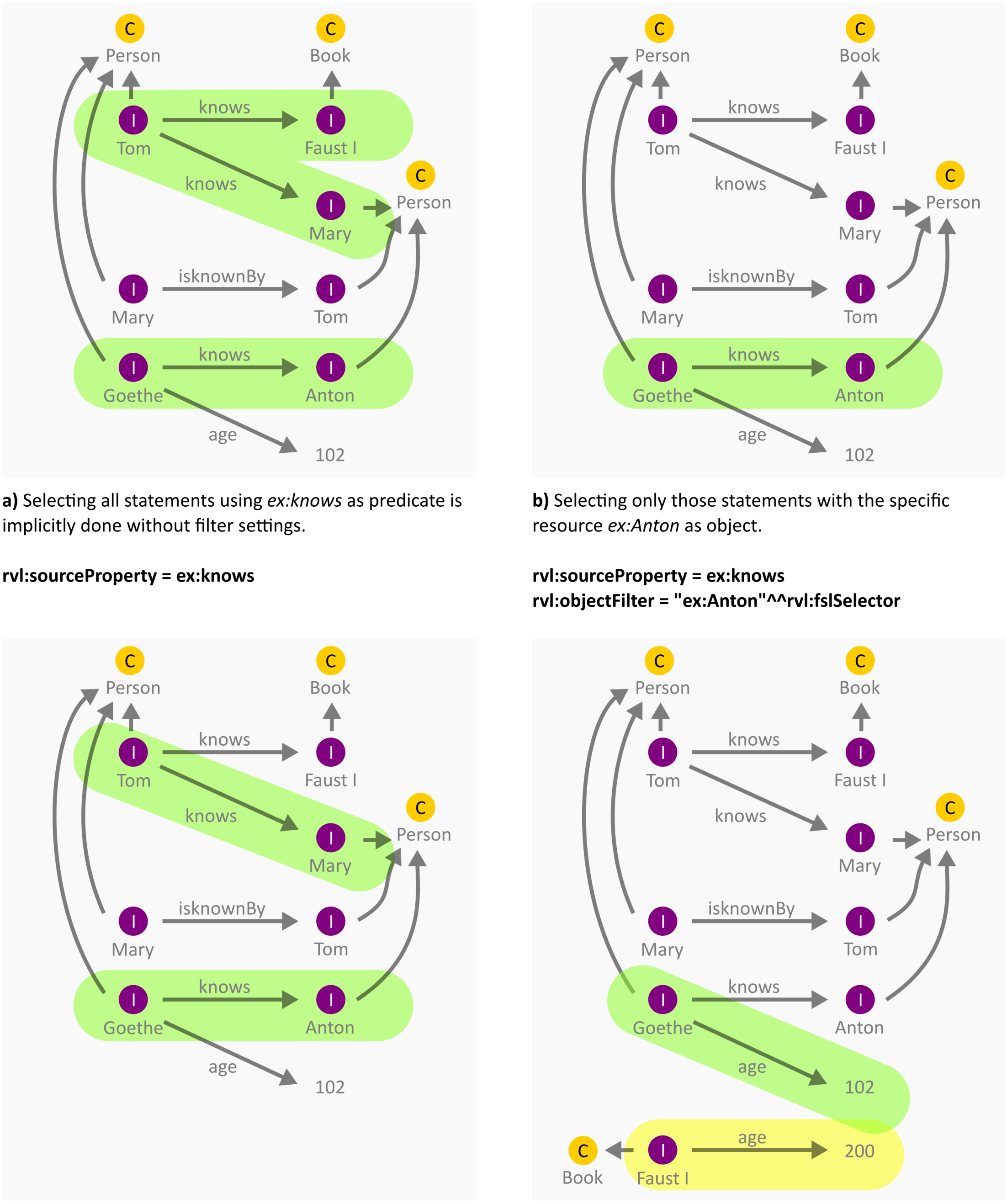
VISO – Comparison of Visualisation Literature

Author(s)	Da	Do	Vo	Re	Ac	Us	Sy	
Allen [All97]						×		Terminology
Amar et al. [AES05]					×			Terminology
Amar, Stasko [AS04]					×			Terminology
Andrienko, Andrienko [AA06]	×				×			Terminology
Bernhard et al. [BDW05]		×		×				Taxonomy
Bertin [Ber83]	×		×	×				Terminology
Brodlie, Noor [BN07]	×							Taxonomy
Burkhard [Bur04]		×		×		×		Taxonomy
Card, Mackinlay [CM97]	×		×					Taxonomy
Card et al. [CMS99]	×		×	×	×	×		Taxonomy
Chi [Chi00]				×	×			Taxonomy
Chuah, Roth [CR96]					×			Taxonomy
Daassi et al. [DNF05]				×				Taxonomy
Duke et al. [DBD04]	×			×	×		×	Ontology
Engelhardt [vE02]			×	×				Taxonomy
Espinosa [EHG99]		×		×	×			Terminology
Fikkert et al. [FDBJK07]					×		×	Terminology
Foni et al. [FPMT10]		×		×				Taxonomy
Freitas et al. [FLC$^+$02]				×	×	×		Taxonomy
Gilson et al. [GSGC08]	×	×	×	×				Ontology
Heer et al. [HHC$^+$08]	×			×		×	×	Terminology
Herman et al. [HMM00]				×	×			Terminology
Keim [Kei02]	×			×	×			Taxonomy
Lau, Vande Moere [LVM07]	×			×	×			Terminology
Le Grande, Soto [LGS02]	×	×		×				Terminology
Lee et al. [LPP$^+$06]				×	×			Taxonomy
Lengler, Eppler [LE07]		×		×				Taxonomy
Limbourg, Vanderdonckt [LV03]					×			Terminology
Lohse et al. [LBWR94]				×				Taxonomy
Mackinlay [Mac86a]			×					Terminology
Norman [Nor86]					×	×		Terminology
Otjacques et al. [OF05]		×		×				Taxonomy
Pfitzner et al. [PHP03]	×	×		×	×	×	×	Taxonomy
Potter and Wright [PW07]				×			×	Ontology
Qin et al. [QZP03]	×	×		×	×			Taxonomy
Rhodes et al. [RKR06]	×	×		×	×	×	×	Ontology
Risch [Ris08]			×					Terminology
Robertson [Rob91]		×		×				Taxonomy
Shneiderman [Shn96]	×				×			Taxonomy
Shu et al. [SAR08]	×			×			×	Ontology
Spence [Spe07]	×		×	×	×			Terminology
Stolte [Sto03]	×		×	×	×			Taxonomy
Tory, Möller [TM04a]	×			×				Taxonomy
Tory, Möller [TM04b]						×		Terminology
Tweedie [Twe97]	×				×			Taxonomy
Valiati [VPF06]					×			Taxonomy
Wehrend, Lewis [WL90]	×			×	×			Taxonomy
Wilkinson [Wil05]	×		×		×			Terminology
Wiss, Carr [WC98]				×	×			Terminology
Yi et al. [YaKSJ07]					×			Taxonomy
Zhang [Zha96]	×		×		×			Taxonomy
Zhou, Feiner [ZF98]					×			Taxonomy
Zudilova-Seinstra [ZS07]						×		Terminology
count	24	12	11	32	30	10	7	

Table B.1: Studied literature on visualisation. We distinguish the fields of data (Da), domain (Do), graphical vocabulary (Vo), graphic representation (Re), activity (Ac), user (Us), and system (Sy). The last column states the level of formalisation (terminology, taxonomy or ontology). This table has been published before at http://mmt.inf.tu-dresden.de/VO.

Appendix C

RVL



a) Selecting all statements using *ex:knows* as predicate is implicitly done without filter settings.

rvl:sourceProperty = ex:knows

b) Selecting only those statements with the specific resource *ex:Anton* as object.

rvl:sourceProperty = ex:knows
rvl:objectFilter = "ex:Anton"^^rvl:fslSelector

c) Selecting only those statements whose objects are typed *ex:Person*.

rvl:sourceProperty = ex:knows
rvl:objectFilter = "rdf:type/ex:Person"^^rvl:fslSelector

d) Distinguishing the type of the *subjects* (e.g., for selecting different visual means for showing the age of a person vs. the age of a book or a tree)

rvl:sourceProperty = ex:age
rvl:subjectFilter = "rdf:type/ex:Person"^^rvl:fslSelector
rvl:subjectFilter = "rdf:type/ex:Book"^^rvl:fslSelector

Figure C.1: Mappings in RVL can be refined with filters: An implicit filtering to statements having the predicate defined by rvl:sourceProperty is done for all property mappings (a). This can be refined by rvl:objectFilter (b,c) and rvl:subjectFilter (d). Filter expressions can be SPARQL or (as shown here) FSL selectors (Sect. 4.3.1). Constraining the mapping to statements with a specific object value (b) is similar to what can be achieved using rvl:ValueMapping. However, a value mapping only ensures that selected source values will be mapped to selected graphic values while it does not limit the property mapping to a subset of statements.

Appendix D

RVL Example Mappings and Application

D.1 Listings of RVL Example Mappings as Required by Prototype P3

While the mappings given in Chapter 9 are complete with respect to the defined visual mappings, sometimes additional settings are necessary, because the current implementation (P3) has a limited support of default settings or styles. In the following, we give the listings for the example mappings *RO-4b* und *CIT-5*, including these additional settings.

```
:RO-4b
  a rvl:PropertyMapping;
  rvl:subjectFilter "http://purl.org/ro/ont#Requirement"^^rvl:classSelector;
  rvl:sourceProperty rdf:type;
  rvl:targetAttribute vg:color_hsl_lightness;
  rvl:valueMapping [
    rvl:sourceValueOrderedSet (
      ro:LowPriority
      ro:MediumPriority
      ro:HighPriority
    );
    rvl:discretize "true";
    rvl:targetValueInterval [
      a rvl:Interval;
      rvl:lowerBoundIncl "0";
      rvl:upperBoundIncl "100";
    ];
  ].


:ShapeMapping
  a rvl:PropertyMapping;
  rvl:sourceProperty rdf:type;
  rvl:subjectFilter "http://purl.org/ro/ont#Requirement"^^rvl:classSelector;
  rvl:targetAttribute vg:shape_named;
  rvl:valueMapping [
    a rvl:ValueMapping;
    rvl:sourceValue ro:Requirement;
    rvl:targetValue common-shapes:Square;
  ].

:LabelMapping
  a rvl:PropertyToGraphicObjToObjRelationMapping;
  rvl:sourceProperty rdf:ID;
  rvl:subjectFilter "http://purl.org/ro/ont#Requirement"^^rvl:classSelector;
  rvl:targetObjToObjRelation vg:Labeling_Relation;
  rvl:subMapping [
    rvl:subMapping-onRole vg:labeling_label;
    rvl:subMapping-onTriplePart rdf:subject;
    rvl:subMapping-mapping :LabelTextIdentity;
  ],[
    rvl:subMapping-onRole vg:this;
    rvl:subMapping-onTriplePart rdf:subject;
    rvl:subMapping-mapping :TypeToLabelStyle;
  ],[
    rvl:subMapping-onRole vg:labeling_base;
    rvl:subMapping-onTriplePart rdf:subject;
    rvl:subMapping-mapping :Type2Width;
  ].

:LabelTextIdentity
  rvl:disabled "true";
  a rvl:IdentityMapping;
  rvl:sourceProperty rvl:label;
  rvl:targetAttribute vg:text_value.

:TypeToLabelStyle
  a rvl:PropertyMapping;
  rvl:disabled "true";
  rvl:sourceProperty rdf:type ;
  rvl:targetAttribute vg:labeling_attachedBy;
  rvl:valueMapping [
    rvl:sourceValue ro:Requirement;
    rvl:targetValue vg:Containment_Relation;
  ].

:Type2Width
  a rvl:PropertyMapping;
  rvl:disabled "true";
  rvl:sourceProperty rdf:type;
  rvl:targetAttribute vg:width;
  rvl:valueMapping [
    a rvl:ValueMapping;
    rvl:sourceValue ro:Requirement;
    # create some space for contained labels:
    rvl:targetValue "100"^^xsd:float;
  ].
```

Listing D.1: RO-4b – RVL mappings.

```
:CIT-5
  a rvl:PropertyMapping;
  rvl:sourceProperty cito:cites;
  rvl:targetObjToObjRelation vg:Linking_Directed_Relation;
  rvl:subMapping [
    rvl:subMapping-onRole vg:linking_connector;
    rvl:subMapping-onTriplePart rdf:predicate;
    rvl:subMapping-mapping :IconLabelMapping;
  ].

:IconLabelMapping
  a rvl:PropertyMapping ;
  rvl:disabled "true";
  rvl:sourceProperty rdf:ID;
  rvl:targetObjToObjRelation vg:Labeling_Relation;
  rvl:subMapping [
    rvl:subMapping-onRole vg:labeling_label;
    rvl:subMapping-onTriplePart rdf:predicate;
    rvl:subMapping-mapping :IconLabelShapeMapping;
  ],[
    rvl:subMapping-onRole vg:labeling_label;
    rvl:subMapping-onTriplePart rdf:predicate;
    rvl:subMapping-mapping :ID2ColorNamed;
  ].
```

```
:IconLabelShapeMapping
  a rvl:PropertyMapping;
  rvl:disabled "true";
  rvl:sourceProperty rdf:ID;
  rvl:targetAttribute vg:shape_named;
  rvl:valueMapping [
    rvl:sourceValueOrderedSet (
      cito:critiques
      cito:confirms );
    rvl:targetValueList (
      common-shapes:XMark
      common-shapes:Star18 );
  ].

# only necessary, since the icons we use here are not yet colored
:ID2ColorNamed
  a rvl:PropertyMapping ;
  rvl:disabled "true" ;
  rvl:sourceProperty rdf:ID ;
  rvl:targetAttribute vg:color_named;
  rvl:valueMapping [
    rvl:sourceValueOrderedSet (
      cito:critiques
      cito:confirms );
    rvl:targetValueList (
      vg:Red
      vg:Yellow );
  ].
```

Listing D.2: CIT-5 – RVL mappings.

D.2 Features Required for Implementing all Sketches

In summary, the following features need to be implemented in order to completely reproduce the subset of sketches from Sect. 3.6:

- Dereference URIs of symbols to allow for arbitrary SVG graphics as shapes including marker shapes, e. g., arrow heads (AA-4/CIT-5/RO-6).
- Improved positioning of labels (RO-5).
- Generate legends (RO-5).
- Defaults – for example, adapt the shape width to the width of labels if the labels are attached by containment, i. e., placed »inside« the shape (PO-9).
- Reflexive edges cannot yet be rendered (not covered by the sketches).

The following features also require conceptual work, e. g., changes or extensions to the RVL specification:

- For handling text values and units such as cost and currency literals, a concept for concatenating texts is needed. Since handling text touches existing technologies like Fresnel, the focus was not yet on this problem (RO-5).
- Set the marker width to line width by default for specific graphic representation types (PO-9).
- Support further graphic representation types such as *indented list*. As a concept for describing indented lists, multiple options exist – e. g., the composition of ordered line-up and position. Similarly, a concept for describing tables is required. Describing tables solely as a product of combining graphic-relations can be challenging. Again multiple options exist according to Engelhardt [vE02] (PO-7).

- A concept of name-value pairs to simplify frequently used label positioning tasks: Name-value pairs are frequently used in the context of labelling (e. g., for the representation of attributes in UML-like diagrams). Both key and value may be represented as graphic or as text (RO-6).
- A concept of rendering the combined usage of containment and linking – including the identification and handling of impossible composition cases. Using multiple *graphic object-to-object relations* that are not orthogonal in the same graphic, requires calculating whether the combination is possible for the concrete given instance data. Here, algorithms from the graph topology could help to spot those cases that allow for composition and those that do not (similar to planarity checks). The AVM is a rich enough model to support running such checks (RO-7).
- The colouring of arrow heads is not yet possible. One option could be to use more fine-grained roles to allow for referencing such parts. A workaround is to define an arrow style that determines the colour of the head (RO-6).

In order to specify and implement not only the subset of sketches from Sect. 3.6, but all sketches presented in Appendix A, some additional features are required. In Table D.1, we give an overview of the overall set of features and the current status of implementation in prototype P3 as well as the current status of specification in RVL, VISO and the AVM. Some features are not intended to be specified in one of the vocabularies. These are marked as N/A. In some cases there is an implementation by prototype P3, but no specification. This mainly applies to features concerning style and defaults, which need to be properly described in the RVL specification.

Feature	Implemented in P3	Specified
Property mapping		
Position	×	×
Relative position	-	-
Relative distance	×	×
Size	×	×
Width	×	×
Shape (incl. complex shapes, consisting of multiple objects)	×	×
Arbitrary shapes / dereference SVG shapes from URL	-	×
Colour (named) for all CSS colour names should be supported	(×)	(×)
Colour HSL lightness	×	×
Linking	×	×
Directed linking	×	×
Undirected linking	×	×
Reflexive edges	-	N/A
Line-up		
Ordered line-up	-	-
Unordered line-up	-	-
Containment		
Containment for arbitrary shapes	-	×
Containment for circles	×	×
Adjacency	-	-
Multipanels	-	N/A
Display partial graphics	-	N/A
Complex (composed) graphic relations		
Builds-on	-	-
Plane connectors	-	-
Co-highlighting	(×)	×
Composed shapes	-	-
Composition of linking and containment	-	(×)
„Satellite“ relation	-	-
Graphic representation types		
Indented list	-	(×)
Table		
Tabular representation with ordinal/nominal axis (or alternatively support a composition of position/line-up and containment)	-	(×)
Colouring of table cells / container background	-	(×)
Inherit property mappings	×	×
Submapping	×	×
Resource mapping	×	×
Value mapping		
Explicit value mappings	×	×
Calculation of value mappings	×	×
Continuous to continuous	×	×
Discretisation	×	×
Continuous to nominal	×	×
Nominal to nominal	×	×
Ordinal to continuous	×	×
Legends for property mappings and value mappings	-	×
Axes	-	-
Labelling		
Support multiple labels	×	×
Support labels that are name-value pairs (textual or iconic)	-	(×)
Of connectors	×	×
Of nodes	×	×
Of labels	×	×
Of graphic objects with arbitrary roles	-	×
Icon labelling	×	×
Text labelling	×	×
Attaching labels via ...		
Superimposition	×	×
Containment (breaking the label text to the shape of the container)	×	×
Use a table for multiple contained labels	-	-
Alignment	(×)	×
Layouting		
Graph layout (e. g., force-driven)	×	N/A
DAG layout	-	N/A
Tree layout	×	N/A
Treefication	×	-
Concat values and units	-	-
Style settings and defaults	×	-
Shadows	×	N/A
Set connector width (and marker width) to node width	-	-
Markers shape	×	-
Default colour	×	-
Default labelling	×	-
Centre text, if label positioned centered	×	-
Invert text label colour by default	×	-
Default marker shape	×	-
Marker width adapts to line width by default	-	-
Highlight neighbour nodes by default	×	-
Highlight duplicated graphic objects by default	×	-
Width adapts to label width by default	-	-
Container width adapts to containees by default	-	-
Amount of implemented/specified features	57% (and additional 4% partially)	53% (and additional 9% partially)

Table D.1: Features required to realise the case studies' sketches as implemented by prototype P3, respectively specified/conceptualised in RVL, VISO and the AVM.

D.3 JSON Format for Processing the AVM with D3 – Hierarchical Variant

```
{
  "children":[
    {
      "uri":"http://purl.org/rvl/example-data/Some_URI_of_some_resource",
      "roles":[
        "containment_container"
        ],
      "shape":"circle",
      "width":17.0,
      "labels":[
        {
          "text_value":"Some URI of some resource",
          "width":8.5,
          "position":"centerRight",
          "type":"text_label"
        }
      ],
      "type":"Containment",
      "children":[
        {
          "uri":"http://purl.org/rvl/example-data/Another_URI_of_another_resource",
          "roles":[
            "containment_containee"
            ],
          "color_rgb_hex_combined":"#ff9900",
          "shape":"circle",
          "width":17.0
        }
      ]
    }
  ],
  "graphic_type":"circle-packing-zoomable"
}
```

Listing D.3: Example of the hierarchical variant of the JSON format used for the communication between the RVL server and the OGVIC D3 plugins. Hierarchically structured data is, for instance, expected by the D3 layouts *tree* and *pack*, which are applied by the graphic types *Collapsible Tree*[1] and *Zoomable Circle Packing*[2] that we use in prototype P3 (cf. Sect. 8.5).

[2] Based on an example from http://bl.ocks.org/mbostock/4339083/, accessed: 29.04.2016

[2] Based on an example from http://bl.ocks.org/mbostock/7607535/, accessed: 29.04.2016

Bibliography

[AA06] N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer, 2006. 104, 107, 113, 114, 230

[Abi97] S. Abiteboul. Querying semi-structured data. *Database Theory – ICDT '97*, pages 1–18, 1997. 17

[ABK$^+$07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proc.*, volume 4825 of *LNCS*, chapter DBpedia: A Nucleus for a Web of Open Data, pages 722–735. Springer, Berlin, Heidelberg, 2007. 20

[Ack89] R. L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, 16(1):3–9, 1989. 17

[ADD10] S. Auer, R. Doehring, and S. Dietzold. Less – template-based syndication and presentation of linked data. *The Semantic Web: Research and Applications*, pages 211–224, 2010. 44, 75

[AES05] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *Proc. of the IEEE Symposium on Information Visualization, 2005 (InfoVis 2005)*, pages 111–117, 2005. 230

[AEWW13a] U. Aßmann, J. Ebert, T. Walter, and C. Wende. *Ontology-Driven Software Development*, chapter Ontology and Bridging Technologies, pages 179–192. Springer, 2013. 9

[AEWW13b] U. Aßmann, J. Ebert, T. Walter, and C. Wende. *Ontology-Driven Software Development*, chapter Ontology-Driven Metamodelling for Ontology-Integrated Modelling, pages 257–274. Springer, Berlin, Heidelberg, 2013. 95

[All97] R. B. Allen. *Handbook of Human-Computer Interaction*, chapter Mental Models and User Models, pages 49–63. Elsevier Science Inc., New York, NY, USA, 1997. 230

[AS04] R. Amar and J. Stasko. A knowledge task-based framework for design and evaluation of information visualizations. In *Proc. of IEEE Symposium on Information Visualization*, pages 143–150, 2004. 230

[AZW06] U. Aßmann, S. Zschaler, and G. Wagner. Ontologies, metamodels, and the model-driven paradigm. *Ontologies for Software Engineering and Software Technology*, 2006. 92

[Aß03a] U. Aßmann. Automatic roundtrip engineering. *Electronic Notes in Theoretical Computer Science*, 82(5):33–41, 2003. SC 2003, Workshop on Software Composition (Satellite Event for ETAPS 2003). 140, 215

[Aß03b] U. Aßmann. *Invasive Software Composition*. Springer, 2003. 54

[BAB$^+$93] D. M. Butler, J. C. Almond, R. D. Bergeron, K. W. Brodlie, and R. B. Haber. Visualization reference models. In *Proc. of the 4th conference on Visualization '93*, pages 337–342, San Jose, California, 1993. IEEE Computer Society. 13

[BANE15] T. Bosch, E. Acar, A. Nolle, and K. Eckert. The role of reasoning for RDF validation. In *Proc. of the 11th International Conference on Semantic Systems*, pages 33–40. ACM, 2015. 174

[BB08] C. Becker and C. Bizer. DBpedia Mobile: A location-aware Semantic Web client. In *Proc. of the Semantic Web Challenge at ISWC 2008*, 2008. 74

[BB10] B. Braatz and C. Brandt. How to modify on the Semantic Web? In *Current Trends in Web Engineering*, volume 6385, pages 187–198. Springer, Berlin, Heidelberg, 2010. 192

[BCE+92] K. W. Brodlie, L. Carpenter, R. A. Earnshaw, J. R. Gallop, R. J. Hubbold, A. M. Mumford, C. D. Osland, and P. Quarendon. *Scientific Visualization: Techniques and Applications*. Springer, 1992. 102

[BDD+04] K. W. Brodlie, D. A. Duce, D. J. Duke, et al. Visualization ontologies: Report of a workshop held at the national e-Science centre. Technical report, e-Science Institute, Edinburgh, Scotland, 2004. 102

[BDW05] J. Bernhard, M. Dragan, and S. Wenzel. Evaluation und Erweiterung der Kriterien zur Klassifizierung von Visualisierungsverfahren für GNL. Technical Report 05001, Fraunhofer IML and Universität Kassel, 2005. 230

[Ber83] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983. 107, 115, 230

[BGH+08] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *Proc. of the International Conference on Web Search and Web Data Mining (WSDM08)*, pages 33–44. ACM, 2008. 22

[BH09] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009. 42, 50, 65, 68, 69, 143

[BH10] M. Brophy and J. Heflin. OWL-PL: a presentation language for displaying semantic data on the web. Technical Report LU-CSE-09-002, Lehigh University, 2010. 75, 137, 139

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked data – the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009. 20

[BK05] J. Bézivin and I. Kurtev. Model-based technology integration with the technical space concept. In *Proc. of the Metainformatics Symposium*. Springer, 2005. 103

[BK12] N. Barnickel and J. Klessmann. *Open Initiatives: Offenheit in der digitalen Welt und Wissenschaft*, chapter Open Data – Am Beispiel von Informationen des öffentlichen Sektors, pages 127–158. universaar, 2012. 20

[BLP05] C. Bizer, R. Lee, and E. Pietriga. Fresnel – display vocabulary for RDF. User Manual. http://www.w3.org/2005/04/fresnel-info/manual/, 2005. 73

[BM04] P. V. Biron and A. Malhotra. XML Schema part 2: Datatypes second edition. W3C Recommendation. http://www.w3.org/TR/xmlschema-2/, 2004. 115

[BN07] K. Brodlie and N. M. Noor. Visualization notations, models and taxonomies. *EG UK Theory and Practice of Computer Graphics*, pages 207–212, 2007. 103, 230

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. 43

[Bor97] W. N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse.* PhD thesis, Universiteit Twente, Netherlands, 1997. 19

[BRR03] N. Boukhelifa, J. Roberts, and P. Rodgers. A coordination model for exploratory multi-view visualization. In *Proc. of the conference on Coordinated and Multiple Views In Exploratory Visualization*, pages 76–85. IEEE, 2003. 12

[BTMS99] R. Bardohl, G. Taentzer, M. Minas, and A. Schürr. *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*, volume 2, chapter Application of Graph Transformation to Visual Languages, pages 105–180. World Scientific, 1999. 139, 192

[Bul08] R. I. Bull. *Model driven visualization: towards a model driven engineering approach for information visualization.* PhD thesis, University of Victoria, 2008. 14, 22, 36, 41, 48, 51, 53, 58, 71, 193

[Bun97] P. Buneman. Semistructured data. In *Proc. of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '97)*, pages 117–121, 1997. 17

[Bur92] S. Burbeck. Applications programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC). http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html, accessed: 12.07.2016, 1992. 14, 41

[Bur04] R. A. Burkhard. Learning from architects: the difference between knowledge visualization and information visualization. In *Proc. of the Eighth International Conference on Information Visualisation*, pages 519–524. IEEE, 2004. 230

[CCKT83] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. Graphical methods for data analysis. *Duxbury Press, Boston, Massachusetts*, 1983. 117, 119

[CDC$^+$07] M. Cammarano, X. Dong, B. Chan, J. Klingner, J. Talbot, A. Halevy, and P. Hanrahan. Visualization of heterogeneous data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1200–1207, 2007. 45

[CE00] K. Czarnecki and U. W. Eisenecker. *Generative Programming–Methods, Tools, and Applications*. Addison-Wesley, 2000. 169

[CEH$^+$09] M. Chen, D. Ebert, H. Hagen, R. S. Laramee, R. van Liere, K. L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver. Data, information, and knowledge in visualization. *Computer Graphics and Applications*, 29(1):12–19, 2009. 17

[Cha09] P. A. Champin. Tal4Rdf: lightweight presentation for the Semantic Web. In *Proc. of the 5th Workshop on Scripting and Development for the Semantic Web (ESWC2009)*, 2009. 75, 87

[Che76] P. P.-S. Chen. The entity-relationship model – toward a unified view of data. *Transactions on Database Systems (TODS)*, 1(1):9–36, 1976. 131

[Chi00] E. H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proc. of the IEEE Symposium on Information Visualization*, pages 69–75, 2000. 12, 13, 230

[Chr95] N. R. Chrisman. Beyond Stevens: a revised approach to measurement for geographic information. In *Proc. of the International Symposium on Computer-Assisted Cartography*, 1995. 113

[CM84] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. 107, 115, 118

[CM97] S. Card and J. Mackinlay. The structure of the information visualization design space. volume 0, page 92, Los Alamitos, CA, USA, 1997. IEEE. 52, 230

[CMS99] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. 12, 13, 113, 230

[CMS09] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, chapter Information visualization. CRC Press, 2009. 13, 67, 105, 109

[CR96] M. C. Chuah and S. F. Roth. On the semantics of interactive visualizations. In *Proc. of the IEEE Symposium on Information Visualization*, pages 29–36. IEEE, 1996. 230

[CTW$^+$09] B. Chan, J. Talbot, L. Wu, N. Sakunkoo, M. Cammarano, and P. Hanrahan. Vispedia: on-demand data integration for interactive visualization and exploration. In *Proc. of the 35th SIGMOD international conference on Management of data*, pages 1139–1142, Providence, Rhode Island, USA, 2009. ACM. 1, 3, 45

[CWT$^+$08] B. Chan, L. Wu, J. Talbot, M. Cammarano, and P. Hanrahan. Vispedia: Interactive visual exploration of wikipedia data via search-based integration. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1213–1220, 2008. 45

[DAR06] S. Dietzold, S. Auer, and T. Riechert. Kollaborative Wissensarbeit mit OntoWiki. In *GI Jahrestagung*, pages 498–508, 2006. 184

[DBD04] D. Duke, K. Brodlie, and D. Duce. Building an ontology of visualization. In *Proc. of the IEEE Conference on Visualization*. IEEE, IEEE, 2004. 102, 230

[DBDH05] D. Duke, K. Brodlie, D. Duce, and I. Herman. Do you see what I mean? [Data visualization]. *Computer Graphics and Applications, IEEE*, 25(3):6–9, 2005. 101

[DNF05] C. Daassi, L. Nigay, and M.-C. Fauvet. A taxonomy of temporal data visualization techniques. *Information-Interaction-Intelligence*, 5(2):41–63, 2005. 230

[DOD06] R. Delbru, E. Oren, and S. Decker. Automatic facet construction from Semantic Web data. In *Proc. of the 29th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval, 2006 Seattle, WA, USA*. ACM, 2006. 92

[EHG99] O. Espinosa, C. Hendrickson, and J. Garrett, J.H. Domain analysis: a technique to design a user-centered visualization framework. In *Proc. of the IEEE Symposium on Information Visualization (Info Vis '99)*, pages 44–52, 1999. 230

[FBGS09] S. M. Falconer, R. I. Bull, L. Grammel, and M. A. Storey. Creating visualizations through ontology mapping. In *Proc. of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '09)*, pages 688–693. IEEE, 2009. 41, 42

[FDBJK07] W. Fikkert, M. D'Ambros, T. Bierz, and T. J. Jankun-Kelly. Interacting with visualizations. In *Human-Centered Visualization Environments*, volume 4417 of *LNCS*, pages 77–162. Springer, 2007. 230

[Few04] S. Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten.* Analytics Press, 2004. 115

[Few09] S. Few. *Now You See It: Simple Visualization Techniques for Quantitative Analysis.* Analytics Press, 2009. 48

[FGJ97] M. Fernandez, A. Gomez-Perez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *Proc. of the Spring Symposium Series on Ontological Engineering*, pages 33–40. American Asociation for Artificial Intelligence, 1997. 100

[FLB$^+$06] T. Furche, B. Linse, F. Bry, D. Plexousakis, and G. Gottlob. RDF querying: Language constructs and evaluation methods compared. In *Reasoning Web*, volume 4126 of *LNCS*, pages 1–52. Springer, 2006. 75

[FLC$^+$02] C. M. D. S. Freitas, P. R. G. Luzzardi, R. A. Cava, M. A. Winckler, M. S. Pimenta, and L. P. Nedel. Evaluating usability of information visualization techniques. In *Proc. of the 5th Workshop on Human Factors in Computer Systems (IHC 2002)*. Brazilian Computer Society Press, 2002. 230

[FPMT10] A. E. Foni, G. Papagiannakis, and N. Magnenat-Thalmann. A taxonomy of visualization strategies for cultural heritage applications. *Journal on Computing and Cultural Heritage (JOCCH)*, 3(1):1–21, 2010. 230

[FS07] S. Falconer and M. Storey. A cognitive support framework for ontology mapping. In *The Semantic Web*, volume 4825 of *LNCS*, pages 114–127. Springer, 2007. 41

[GDD05] D. Gašević, D. Djurić, and V. Devedžić. Bridging MDA and OWL ontologies. *Journal of Web Engineering*, 4(2):118–143, 2005. 9

[GH05] H. Gassert and A. Harth. From graph to GUI: displaying RDF data from the web with Arago. In *In Proc. of the Workshop on Scripting for the Semantic Web (ESCW 2005)*, 2005. 74

[GHH$^+$07] C. Golbreich, M. Horridge, I. Horrocks, B. Motik, and R. Shearer. OBO and OWL: leveraging semantic web technologies for the life sciences. In *The Semantic Web*, number 4825 in LNCS, pages 169–182. Springer, 2007. 20

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994. 14, 105

[GO94] T. R. Gruber and G. R. Olsen. An ontology for engineering mathematics. In *Proc. of 4th International Conference on Principles of Knowledge Representation and Reasoning (KR '94)*, pages 258–269. Morgan Kaufmann, 1994. 115

[GOS09] N. Guarino, D. Oberle, and S. Staab. What is an ontology? In *Handbook on Ontologies*, International Handbooks on Information Systems, pages 1–17. Springer, 2009. 19

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. 19

[Gru09] T. R. Gruber. "ontology". *Encyclopedia of Database Systems.* Springer, 2009. 19

[GSGC08] O. Gilson, N. Silva, P. W. Grant, and M. Chen. From web data to visualization via ontology mapping. In *Computer Graphics Forum*, volume 27, pages 959–966, 2008. 1, 43, 44, 55, 102, 126, 230

[Gua98] N. Guarino. Formal ontology an information systems. In *Proc. of the First International Conference on Formal Ontology in Information Systems (FOIS '98)*, volume 46. IOS Press, 1998. 58

[HA06] J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, 2006. 14

[Hal01] T. Halpin. Object role modeling: An overview. White paper. http://www.orm.net/, accessed: 12.02.2016., 2001. 129

[Hal05] T. Halpin. ORM 2 Graphical Notation. Technical Report ORM2-01, Neumont University, 2005. 129

[Hau09] M. Hausenblas. Exploiting linked data to build web applications. *IEEE Internet Computing*, 13(4):68–73, 2009. 20

[HB10] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 1149–1156, 2010. 35, 43, 50, 65, 66, 68, 69, 143

[HHC$^+$08] J. Heer, F. Ham, S. Carpendale, C. Weaver, and P. Isenberg. Creation and collaboration: Engaging new audiences for information visualization. In *Information Visualization*, volume 4950 of *LNCS*, pages 92–133. Springer, 2008. 110, 230

[HM90] R. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in Scientific Computing*, pages 74–93, 1990. 12, 13

[HMFK10] F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: a SPARQL-based graphical query language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*, pages 87–116. Springer, 2010. 214

[HMM00] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000. 230

[HPVH03] I. Horrocks, P. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003. 93

[HSM07] P. Hanrahan, C. Stolte, and J. Mackinlay. Visual analysis for everyone: Understanding data exploration and visualization. Whitepaper, Tableau Software Inc., 2007. 65, 67

[Huy07] D. F. Huynh. *User interfaces supporting casual data-centric interactions on the Web*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2007. 35

[HZL08] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A browser for the web of data. In *Proc. of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08)*, Aachen, 2008. CEUR-WS. 5, 170

[JS91] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc. of the IEEE Conference on Visualization (Visualization'91)*, pages 284–291. IEEE, 1991. 106

[Kar13] D. Karger. Standards opportunities around data-bearing web pages. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987), 2013. 17

[Kei02] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002. 48, 110, 114, 230

[KFS08] J. Koch, T. Franz, and S. Staab. Lena-browsing RDF data more complex than FOAF. In *Proc. of the 7th International Semantic Web Conference (ISWC), Demo Session*, 2008. 74

[KHI11] H. Knublauch, J. A. Hendler, and K. Idehen. SPIN – Overview and Motivation. W3C Member Submission. http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/, accessed: 10.02.2016, 2011. 21

[KHL$^+$07] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods – a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007. 1

[Knu10] H. Knublauch. SPARQL Web Pages (SWP). Website. http://www.topquadrant.com/uispin/, accessed: 10.12.2016., 2010. 76

[KPSP83] S. Kosslyn, S. Pinker, W. Simcox, and L. Parkin. *Understanding Charts and Graphs: A Project in Applied Cognitive Science.* ERIC Document Reproduction Service ED, 1983. 117

[KS93] A. Khurshid and H. Sahai. Scales of measurements: an introduction and a selected bibliography. *Quality and Quantity*, 27(3):303–324, 1993. 113

[KS12] A. Kerren and F. Schreiber. Toward the role of interaction in visual analytics. In *Proc. of the 2012 Winter Simulation Conference (WSC'12).* ACM, 2012. 36, 215

[KV98] K. Kemp and A. Vckovski. Towards an ontology of fields. In *Proc. of the 3rd International Conference on GeoComputation.* University of Bristol, 1998. 110, 113, 114

[LBWR94] G. L. Lohse, K. Biolsi, N. Walker, and H. H. Rueter. A classification of visual representations. *Communications of the ACM*, 37(12):36–49, 1994. 230

[LCP$^+$10] C. Letondal, S. Chatty, G. Philips, F. André, and S. Conversy. Usability requirements for interaction-oriented development tools. In *22nd Annual Workshop on the Psychology of Programming Interest Group (PPIG 2010)*, Madrid, Spain, 2010. 193

[LE07] R. Lengler and M. J. Eppler. Towards a periodic table of visualization methods for management. In *Proc. of the Conference on Graphics and Visualization in Engineering (GVE 2007)*, pages 1–6, Clearwater, Florida, USA, 2007. IASTED. 230

[LGS02] B. Le Grand and M. Soto. *Visualizing the Semantic Web: Xml-Based Internet and Information Visualization*, chapter Topic Maps, RDF Graphs, and Ontologies Visualization, pages 59–79. Springer Science & Business Media, London, 2nd edition, 2002. 230

[Lie05] H. W. Lie. *Cascading Style Sheets.* PhD thesis, University of Oslo, Oslo, Norway, 2005. 16, 35, 143

[LNB14] S. Lohmann, S. Negru, and D. Bold. The ProtégéVOWL plugin: Ontology visualization for everyone. In *Proc. of ESWC 2014 Satellite Events*, volume 8798 of *LNCS*, pages 395–400. Springer, 2014. 2

[LNHE14] S. Lohmann, S. Negru, F. Haag, and T. Ertl. VOWL 2: User-oriented visualization of ontologies. In *Proc. of the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW '14)*, volume 8876 of *LNAI*, pages 266–281. Springer, 2014. 1, 2

[LNS06] S. Lange, T. Nocke, and H. Schumann. Visualisierungsdesign – ein systematischer Überblick. *Proc. SimVis' 06*, pages 113–128, 2006. 2, 14

[LPP$^+$06] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proc. of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV '06)*, pages 1–5, New York, NY, USA, 2006. ACM. 230

[LV03] Q. Limbourg and J. Vanderdonckt. Comparing task models for user interface design. In D. Diaper and N. Stanton, editors, *The handbook of task analysis for human-computer interaction*, chapter Comparing Task Models for User Interface Design, pages 135–154. Lawrence Erlbaum Associates, 2003. 230

[LVJ12] M. Lenk, A. Vitzthum, and B. Jung. Model-driven iterative development of 3d web-applications using SSIML, X3d and JavaScript. In *Proc. of the 17th International Conference on 3D Web Technology*, pages 161–169. ACM, 2012. 215

[LVM07] A. Lau and A. Vande Moere. Towards a model of information aesthetics in information visualization. In *Proc. of the 11th International Conference on Information Visualization (IV '07)*, pages 87–92, Washington, DC, USA, 2007. IEEE. 230

[Mac86a] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986. 12, 13, 14, 52, 65, 97, 99, 101, 103, 104, 105, 107, 115, 116, 117, 118, 119, 121, 123, 126, 153, 230

[Mac86b] J. D. Mackinlay. *Automatic Design of Graphical Presentations.* PhD thesis, Stanford Univ., CA, USA, 1986. 118

[Mar74] L. E. Marks. *Sensory Processes: The New Psychophysics.* Academic Press, New York, NY, USA, 1974. 113

[Maz09] R. Mazza. *Introduction to information visualization.* Springer, 2009. 107, 113, 114

[MHK10] J. Masters, R. Hodgson, and P. J. Keller. QUDT – quantities, units, dimensions and types. http://www.qudt.org/, accessed: 12.02.2016., 2010. 115

[MHS07] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007. 54, 65

[Mil13] J. J. Miller. Graph Database Applications and Concepts with Neo4j. In *Proc. of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, 2013. 214

[Min00] M. Minas. Hypergraphs as a uniform diagram representation model. *Theory and Application of Graph Transformations*, pages 405–411, 2000. 136, 139, 140, 214

[MKSE11] R. Müller, P. Kovacs, J. Schilbach, and U. Eisenecker. Generative software visualization: Automatic generation of user-specific visualisations. In *2nd International ACM/GI Workshop on Digital Engineering (IWDE)*, pages 45–49, 2011. 42

[MWa] ”guidance”, ”guide”. *Merriam-Webster.com.* Merriam-Webster, 2016. Web. 12.2.2016. 22

[Nar96] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction.* The MIT Press, 1996. 126

[Nor86] D. A. Norman. Cognitive engineering. In *User Centered System Design – New Perspectives on Human-Computer Interaction*, pages 31–61. Lawrence Erlbaum, Hillsdale, NJ, USA, 1986. 230

[OAS09] OASIS quantities and units of measure ontology standard (QUOMOS) TC. OASIS QUOMOS TC. Website of the Technical Committee (closed by 13.08.2014). https://www.oasis-open.org/committees/quomos/, accessed: 12.02.2016., 2009. 115

[OBO] The open biological and biomedical ontologies. http://obofoundry.org/, accessed: 12.02.2016. 20

[ODD06] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for RDF data. In *The Semantic Web – ISWC 2006*, volume 4273 of *LNCS*, pages 559–572. Springer, 2006. 5, 36, 92, 170

[OF05] B. Otjacques and F. Feltz. Characterizing the visualization techniques of project-related interactions. In *Proc. of the 22nd CIB W78 International Conference on Information Technology for Construction*, volume 11 of *Special Issue Process Modelling, Process Management and Collaboration*, pages 113–120. ITcon, 2005. 230

[OMG09] Ontology Definition Metamodel Version 1.0. Object Management Group. OMG Document. http://www.omg.org/spec/ODM/1.0/, accessed: 12.02.2016., 2009. 95

[OWL04] OWL Web Ontology Language reference. W3C Recommendation. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004. 19, 97

[Pat13] P. Patel. A constraint-based visualisation framework for linked data in the RDF technical space. Master thesis. TU Dresden, Germany, 2013. 163, 184

[PBKL06a] E. Pietriga, C. Bizer, D. Karger, and R. Lee. Fresnel: A browser-independent presentation vocabulary for RDF. Presentation slides use at the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA, 2006. 74, 82, 83

[PBKL06b] E. Pietriga, C. Bizer, D. Karger, and R. Lee. *The Semantic Web – ISWC 2006: Proc. of the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA.*, chapter Fresnel: A Browser-Independent Presentation Vocabulary for RDF, pages 158–171. Springer, 2006. 52, 73

[PENN07] M. Palmér, F. Enoksson, M. Nilsson, and A. Naeve. Annotation profiles: Configuring forms to edit RDF. *International Conference on Dublin Core and Metadata Applications*, pages 10–21, 2007. 94

[PHP03] D. Pfitzner, V. Hobbs, and D. Powers. A unified taxonomic framework for information visualization. In *Proc. of the Asia-Pacific Symposium on Information Visualisation (APVis ’03)*, pages 57–66, Darlinghurst, Australia, 2003. Australian Computer Society, Inc. 230

[Pie05] E. Pietriga. Fresnel Selector Language for RDF. http://www.w3.org/2005/04/fresnel-info/fsl/, accessed:12.12.2015., 2005. 73

[Pie07] E. Pietriga. IsaViz: A visual authoring tool for RDF. http://www.w3.org/2001/11/IsaViz/, accessed: 12.12.2015., 2001–2007. 72

[PMWM08] S. Pietschmann, A. Mitschick, R. Winkler, and K. Meißner. CroCo: ontology-based, cross-application context management. In *Proc. of the 2008 Third International Workshop on Semantic Media Adaptation and Personalization*, pages 88–93, 2008. 126

[Pol13] J. Polowinski. Towards RVL: a declarative language for visualizing RDFS/OWL data. In *Proc. of the 3rd International Conference on Web Intelligence, Mining and Semantics.* ACM, 2013. 6, 141, 166, 211

[Pol15] J. Polowinski. RVL GitHub repository. https://github.com/semvis/rvl/, accessed: 12.2.2016., 2015. 211

[PP10] H. Paulheim and F. Probst. Ontology-enhanced user interfaces: A survey. *International Journal on Semantic Web & Information Systems*, 6(2):36–59, 2010. 1, 92

[Pry75] L. S. Prytulak. Critique of S. S. stevens' theory of measurement scale classification. *Perceptual and Motor Skills*, 41(1):3–28, 1975. 113

[PSA$^+$12] J. Z. Pan, S. Staab, U. Aßmann, J. Ebert, and Y. Zhao. *Ontology-Driven Software Development.* Springer Science & Business Media, 2012. 94, 163

[PV13] J. Polowinski and M. Voigt. VISO: a shared, formal knowledge base as a foundation for semi-automatic InfoVis systems. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI WIP '13, Paris, France, 2013. ACM. 6, 97, 211

[PW07] R. Potter and H. Wright. *Interactive Systems. Design, Specification, and Verification: 13th International Workshop, DSVIS 2006, Dublin, Ireland. Revised Papers*, chapter An Ontological Approach to Visualization Resource Management, pages 151–156. Springer, Berlin, Heidelberg, 2007. 102, 230

[QK05] D. Quan and D. Karger. Xenon: An RDF stylesheet ontology. In *Proc. of the 14th International Conference on World Wide Web (WWW 2005), Chiba, Japan.* ACM, 2005. 75

[QZP03] C. Qin, C. Zhou, and T. Pei. Taxonomy of visualization techniques and systems – concerns between users and developers are different. In *Proc. of the Asia GIS Conference 2003*, 2003. 230

[Rau05] A. Rauschmayer. An RDF editing platform for software engineering. *Workshop on Semantic Web Enabled Software Engineering (SWESE), ISWC 2005*, 2005. 74

[Rau10] A. Rauschmayer. *Connected Information Management.* PhD thesis, LMU München, Germany, 2010. 17, 74

[RCC09] M. Rico, D. Camacho, and O. Corcho. Macros vs. scripting in VPOET. In *Proc. of 5th Scripting for the Semantic Web Workshop at the ESWC (SFSW 2009).* CEUR Workshop Proceedings, 2009. 73, 137

[RDF04a] RDF primer. W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004. 19

[RDF04b] RDF vocabulary description language 1.0: RDF schema. W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 2004. 19, 97

[Ris08] J. S. Risch. On the role of metaphor in information visualization. *arXiv.org*, arXiv:0809.0884, September 2008. 230

[RK08] A. Rauschmayer and M. Kiesel. *Emerging Technologies for Semantic Work Environments: Techniques, Methods, and Applications: Techniques, Methods, and Applications*, chapter Lightweight data modeling in RDF. Premier reference source. IGI Global, 2008. 94

[RKR06] P. Rhodes, E. Kraemer, and B. Reed. VisIOn: An interactive visualization ontology. In *Proc. of the 44th Annual Southeast Regional Conference*, ACM-SE 44, pages 405–410, New York, NY, USA, 2006. ACM. 102, 230

[Rob91] P. K. Robertson. A methodology for choosing data representations. *IEEE Computer Graphics and Applications*, 11(3):56–67, 1991. 230

[RS96] J. Rekers and A. Schürr. A graph based framework for the implementation of visual environments. In *Proc. of the IEEE Symposium on Visual Languages*, pages 148–155. IEEE, 1996. 139

[San04] S. R. d. Santos. *A Framework for the Visualization of Multidimensional and Multivariate Data*. PhD thesis, University of Leeds, England, 2004. 114

[SAR08] G. Shu, N. J. Avis, and O. F. Rana. Bringing semantics to visualization services. *Advances in Engineering Software*, 39(6):514–520, 2008. 102, 230

[SBF98] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998. 19

[SBLH06] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006. 1

[SC02] J. L. Sourrouille and G. Caplat. Constraint checking in UML modeling. In *Proc. of the 14th international conference on Software engineering and knowledge engineering (SEKE '02)*, pages 217–224, New York, NY, USA, 2002. ACM. 172

[Sei03] E. Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, 2003. 21

[Sei11] M. Seifert. *Designing round-trip systems by change propagation and model partitioning*. PhD thesis, Technische Universität Dresden, Germany, 2011. 215

[SES] Sesame. http://rdf4j.org/, accessed: 04.05.2015. 190

[SHA15] SHACL use cases and requirements. W3C First Public Working Draft. http://www.w3.org/TR/2015/WD-shacl-ucr-20150414/, April 2015. 214

[SHA16] Shapes Constraint Language (SHACL). W3C Working Draft. https://www.w3.org/TR/shacl/, January 2016. 214

[Shn96] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. of the IEEE Symposium on Visual Languages*, pages 364–371. IEEE, 1996. 48, 110, 230

[SI90] H. Senay and E. Ignatius. Rules and principles of scientific data visualization. Technical Report GWU-IIST-90-13, Institute for Information Science and Technology, Department of Electrical Engineering and Computer Science, School of Engineering and Applied Science, George Washington University, 1990. 115, 117, 119, 124

[SI94] H. Senay and E. Ignatius. A knowledge-based system for visualization design. *Computer Graphics and Applications, IEEE*, 14(6):36–47, 1994. 115, 119, 126

[SMO$^+$03] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498, 2003. 39

[SMPV10] T. Strobl, M. Minas, A. Pleuss, and A. Vitzthum. From the behavior model of an animated visual language to its editing environment based on graph transformation. In *Proc. of the 17th International Conference on 3D Web Technology*, volume 32 of *Electronic Communications of the EASST*, page 13, 2010. 193

[SPA08] SPARQL Query language for RDF. W3C Recommendation. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, 2008. 20

[SPA13] SPARQL 1.1 Update. W3C Recommendation. http://www.w3.org/TR/2013/REC-sparql11-update-20130321/, 2013. 20

[Spe07] R. Spence. *Information Visualization: Design for Interaction*. Prentice-Hall, Inc., 2nd edition, 2007. 112, 230

[Spl08] A. Splendiani. RDFScape: Semantic Web meets systems biology. *BMC Bioinformatics*, 9(4):1–14, 2008. 39, 146, 149

[SSR98] S. Si-Said and C. Rolland. Formalising Guidance for the CREWS Goal-Scenario Approach to Requirements Engineering. In *Proc. of the European-Japanese Conference on Information Modelling and Knowledge Bases*, pages 1–19, Finland, 1998. Université Panthéon-Sorbonne, Paris, France. 22

[Sta06] J. Stasko. Information visualization. Lecture slides CS 7450. Georgia Institute of Technology, Atlanta, USA, 2006. 48

[Ste46] S. S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, June 1946. 113

[Ste00] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000. 132

[Ste03] D. Steer. TreeHugger 0.1. http://rdfweb.org/people/damian/treehugger/, accessed: 03.01.2011., 2003. 75

[STH02] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002. 38

[Sto03] C. R. Stolte. *Query, Analysis, and Visualization of Multidimensional Databases.* PhD thesis, Stanford University, 2003. 230

[STZ$^+$11] K. Siegemund, E. J. Thomas, Y. Zhao, J. Pan, and U. Aßmann. Towards ontology-driven requirements engineering. In *Proc. of the Workshop on Semantic Web Enabled Software Engineering, 10th International Semantic Web Conference (ISWC), Bonn*, 2011. 3, 27

[Szy97] C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* SEI Series in Software Engineering. ACM Press, 1997. 54

[TAB] Tableau software. http://www.tableausoftware.com, accessed: 12.07.2015. 1

[Tay86] W. R. Taylor. The classification of amino acid conservation. *Journal of Theoretical Biology*, 119(2):205–218, 1986. 26

[TM04a] M. Tory and T. Möller. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):72–84, 2004. 230

[TM04b] M. Tory and T. Möller. Rethinking visualization: A High-Level taxonomy. In *IEEE Symposium on Information Visualization (INFOVIS 2004)*, pages 151–158, 2004. 102, 103, 114, 230

[Tru06] F. Truyen. The fast guide to model driven architecture: The basics of model driven architecture. Whitepaper, Cephas Consulting Corp., 2006. 58, 169

[TT01] B. N. Taylor and A. Thompson. *The international system of units (SI).* US Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2001. 113, 115

[Tuf83] E. R. Tufte. *The visual display of quantitative information*, volume 7. Graphics Press Cheshire, CT, USA, 1983. 12, 115

[Twe97] L. Tweedie. Characterizing interactive externalizations. In *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, pages 375–382, New York, NY, USA, 1997. ACM. 230

[Tü99] C. Türker. *Semantic Integrity Constraints in Federated Database Schemata.* Dissertationen zu Datenbanken und Informationssystemen. Ios PressInc, 1999. 172

[UG04] M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64, 2004. 17, 18, 19

[vE02] J. von Engelhardt. *The Language of Graphics.* PhD thesis, Institute for Logic, Language & Computation, University of Amsterdam, 2002. 6, 30, 52, 99, 104, 105, 107, 109, 113, 120, 132, 133, 178, 230, 235

[Ven48] F. A. Veniar. Difference Thresholds for Shape Distortion of Geometrical Squares. *The Journal of Psychology*, 26(2):461–476, 1948. 119

[VP11] M. Voigt and J. Polowinski. Towards a unifying visualization ontology. Technical Report TUD-FI11-01, TU Dresden, Institut für Software und Multimediatechnik, Dresden, Germany, 2011. ISSN: 1430-211X. 6, 97, 126, 211

[VPF06] E. R. A. Valiati, M. S. Pimenta, and C. M. D. S. Freitas. A taxonomy of tasks for guiding the evaluation of multidimensional visualizations. In *Proc. of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV '06)*, BELIV '06, pages 1–6, New York, NY, USA, 2006. ACM. 230

[VPGM12] M. Voigt, S. Pietschmann, L. Grammel, and K. Meißner. Context-aware recommendation of visualization components. In *Proc. of the 4th International Conference on Information, Process, and Knowledge Management (eKNOW 2012)*. XPS, 2012. 127, 212

[VPM13] M. Voigt, S. Pietschmann, and K. Meißner. A semantics-based, end-user-centered information visualization process for semantic web data. In *Semantic Models for Adaptive Interactive Systems*, pages 83–107. Springer, 2013. 1, 3, 212

[Vra12] D. Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proc. of the 21st international conference companion on World Wide Web*, pages 1063–1064. ACM, 2012. 214

[VW93] P. F. Velleman and L. Wilkinson. Nominal, ordinal, interval, and ratio typologies are misleading. *American Statistician*, 47(1):65–72, 1993. 112

[VWPM12] M. Voigt, A. Werstler, J. Polowinski, and K. Meißner. Weighted faceted browsing for characteristics-based visualization selection through end users. In *Proc. of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12)*, pages 151–156, New York, NY, USA, 2012. ACM. 5, 92, 170

[Vö05] M. Völkel. Writing the Semantic Web with Java. Technical report, DERI Galway, Ireland, 2005. 190

[Vö06] M. Völkel. RDFReactor – From ontologies to programmatic data access. In *Proc. of the Jena User Conference 2006*. HP Bristol, Online, 2006. 190

[W3C13] Web Style Sheets home page. W3C. http://www.w3.org/Style/, accessed: 12.12.2015., 2013. 15, 16

[Wal03] N. Walsh. RDF twig: accessing RDF graphs in XSLT. In *Proc. of Extreme Markup Languages*. Citeseer, 2003. 75

[War04] C. Ware. *Information visualization: perception for design*. Morgan Kaufmann, 2004. 113

[WC98] U. Wiss and D. Carr. A cognitive classification framework for 3-dimensional information visualization. Technical Report LTU-TR–1998/4–SE, Luleå University of Technology, Sweden, 1998. 230

[WE09] T. Walter and J. Ebert. Combining DSLs and ontologies using metamodel integration. In *Domain-Specific Languages*, pages 148–169. Springer, 2009. 90

[Wil05] L. Wilkinson. *The grammar of graphics*. Springer, 2005. 52, 54, 55, 62, 70, 103, 104, 109, 112, 113, 117, 120, 124, 127, 216, 230

[WL90] S. Wehrend and C. Lewis. A problem-oriented classification of visualization techniques. In *Proc. of the 1st conference on Visualization (VIS '90)*, pages 139–143, Los Alamitos, CA, USA, 1990. IEEE. 230

[WRRN01] L. Wilkinson, M. Rubin, D. Rope, and A. Norton. nViZn: an algebra-based visualization system. In *Proc. of the 1st International Symposium on Smart Graphics*, pages 76–82, 2001. 38

[Wu07] H. Wu. *Grammar-driven generation of domain-specific language testing tools using aspects*. PhD thesis, The University of Alabama, USA, 2007. 67

[WW10] G. Wills and L. Wilkinson. AutoVis: automatic visualization. *Information Visualization*, 9(1):47–69, 2010. 54

[YaKSJ07] J. Yi, Y. ah Kang, J. Stasko, and J. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007. 48, 230

[YWR03] J. Yang, M. O. Ward, and E. A. Rundensteiner. *Data Visualization: The State of the Art*, chapter Hierarchical Exploration of Large Multivariate Data Sets, pages 201–212. Springer, Boston, MA, USA, 2003. 114

[ZF98] M. X. Zhou and S. K. Feiner. Visual task characterization for automated visual discourse synthesis. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 392–399. ACM Press/Addison-Wesley Publishing Co., 1998. 230

[Zha96] J. Zhang. A representational analysis of relational information displays. *International Journal of Human-Computer Studies*, 45(1):59–74, 1996. 230

[Zhu07] Y. Zhu. *Proc. of the conference of Advances in Visual Computing: Third International Symposium (ISVC 2007), Part II*, chapter Measuring Effective Data Visualization, pages 652–661. Springer, 2007. 118

[ZS07] E. Zudilova-Seinstra. On the role of individual human abilities in the design of adaptive user interfaces for scientific problem solving environments. *Knowledge Information Systems*, 13(2):243–270, 2007. 230

List of Figures

1.1 Example of a node-link representation for the Zebrafish Anatomy Ontology . . . 2
1.2 Example of a tailor-made visualisation for the Zebrafish Anatomy Ontology . . . 3
1.3 Motivating example: An iterative, guided visualisation process 4
1.4 The principle of the OGVIC approach 7
1.5 Unique selling points of the OGVIC approach 8

2.1 Example of a pipeline model for visualisation 12
2.2 Visualisation Reference Model 13
2.3 The Reference Model pattern 14
2.4 Continuum of data modelling languages 18

3.1 Identified ontological concepts and relations for visual mapping 28
3.2 First set of sketches for the case studies' ontologies 30
3.3 Second set of sketches for the case studies' ontologies 31
3.4 Third set of sketches for the case studies' ontologies 32
3.5 Sketch of a tabular representation as an example for a composed graphic 33

4.1 Cytoscape – Visual Mapper view 40
4.2 ZEST – Static Graph Viewer 41
4.3 CogZ+MDV – Mapping View 42
4.4 Protovis – Bar chart example 43
4.5 SemViz – Semantic Bridging Ontology 44
4.6 Vispedia – Interactive mapping interface 45
4.7 Overview of visualisation approaches 61
4.8 GPL – Example of a grouped bar graph 64
4.9 Tableau – User interface for VizQL 65
4.10 GSS – Examplary styling result 72
4.11 Fresnel – Foundational concepts 73
4.12 UISPIN – Example 77
4.13 Visual mapping on instance level vs. class level 79
4.14 Extended Editing Framework (EEF) – Example 91
4.15 Graphical Modelling Framework (GMF) – Architecture 91
4.16 TopBraid Composer – Inference engine configuration 96

5.1 VISO modules 98
5.2 Example of using VISO 99
5.3 Summary of visualisation literature reviewed for VISO 102
5.4 VISO/graphic module – Main classes 104
5.5 Graphic representations – Examples 106
5.6 Graphic attributes – Examples 108
5.7 Graphic relations – Overview 111
5.8 VISO/data module – Main classes 112

5.9 Tabular data model . . . 114
5.10 VISO/facts module – Main properties . . . 116
5.11 Cleveland and McGill's ranking of perceptual tasks . . . 118
5.12 Mackinlay's ranking of perceptual tasks . . . 119
5.13 Syntactic interaction . . . 122
5.14 Perceptual interaction . . . 123
5.15 Conflicts in composing graphic objects . . . 125
5.16 Documentation of VISO . . . 128

6.1 Example AVM – Linking (default graphical RDF notation) . . . 130
6.2 Example AVM – Linking (graphical role notation) . . . 130
6.3 Example AVM – Containment relation . . . 131
6.4 Syntactic roles in the work of Engelhardt . . . 133
6.5 Classification of the composition of graphic objects . . . 134
6.6 Example AVM – Two graphic relations used in combination . . . 134

7.1 Principle of RVL . . . 142
7.2 RVL – Main classes . . . 145
7.3 rvl:PropertyMapping – Overview . . . 146
7.4 Property mappings – Examples . . . 147
7.5 rvl:ValueMapping – Overview . . . 150
7.6 Calculation of value mappings . . . 151
7.7 Determining the scale of measurement of source values (decision diagram) . . . 155
7.8 Determining source values addressed by a value mapping (decision diagram) . . . 156
7.9 Determining target values addressed by a value mapping (decision diagram) . . . 157
7.10 Submappings – Example . . . 161

8.1 Architecture OGVIC visualisation design system . . . 171
8.2 Effectiveness constraint – Warning . . . 176
8.3 Effectiveness constraint – Quickfix . . . 177
8.4 Simultaneous composition – Confluence . . . 179
8.5 Prototype P1 – Architecture . . . 183
8.6 Prototype P1 – Bounds constraint . . . 183
8.7 Prototype P2 – Architecture . . . 185
8.8 Prototype P2 – Visualise view . . . 186
8.9 Prototype P2 – Effectiveness warning . . . 186
8.10 Prototype P3 – Detailed architecture and main classes . . . 187
8.11 Prototype P3 – Architecture . . . 189
8.12 Prototype P3 – Web UI . . . 191
8.13 Suggested Model-View-Controller architecture . . . 194
8.14 RVL mapping editor – Mockup . . . 195

9.1 RVL example RO-4b . . . 197
9.2 RVL example CIT-1 . . . 198
9.3 RVL example CIT-5 . . . 199
9.4 RVL example AA-3 . . . 200
9.5 RVL example AA-4 . . . 201
9.6 RVL example PO-9 . . . 202
9.7 RVL example PO-5 . . . 204

A.1 Sketches for the case studies' ontologies – Software technology I . . . 220
A.2 Sketches for the case studies' ontologies – Software technology II . . . 221
A.3 Sketches for the case studies' ontologies – Publishing . . . 222
A.4 Sketches for the case studies' ontologies – Plant Ontology I . . . 223

A.5 Sketches for the case studies' ontologies – Plant Ontology II 224
A.6 Sketches for the case studies' ontologies – Zebra Fish Anatomy Ontology I 225
A.7 Sketches for the case studies' ontologies – Zebra Fish Anatomy Ontology II . . . 226
A.8 Sketches for the case studies' ontologies – Amino Acids Ontology 227

C.1 Filters in RVL property mappings – Overview 232

List of Tables

2.1 Comparison of XSL and CSS . . . 16

3.1 Characteristics of relations by occurrence . . . 29

4.1 Comparison of vis. approaches – Data . . . 47
4.2 Comparison of vis. approaches – Visualisation and interaction . . . 49
4.3 Comparison of vis. approaches – Configurability . . . 51
4.4 Comparison of vis. approaches – Explicit presentation definition & variability . . 53
4.5 Comparison of vis. approaches – Composition . . . 54
4.6 Comparison of vis. approaches – Level of automation . . . 55
4.7 Comparison of vis. approaches – Use of data semantics & vis. knowledge . . . 57
4.8 Comparison of vis. approaches – Guidance . . . 58
4.9 Comparison of vis. approaches – Ontology usage . . . 59
4.10 Comparison of vis. languages – Supported data models & constructs . . . 67
4.11 Comparison of vis. languages – Interactivity . . . 68
4.12 Comparison of vis. languages – Language paradigm . . . 68
4.13 Comparison of vis. languages – Platform and visual structure independence . . . 69
4.14 Comparison of vis. languages – Extensibility, shareability and composability . . . 70
4.15 Comparison of vis. languages – Editor construction & user familiarity . . . 71
4.16 Applications using Fresnel. . . . 74
4.17 Comparison of RDF display languages – Supported data models & constructs . . 78
4.18 Comparison of RDF display languages – Visual mapping and interactivity . . . 79
4.19 Comparison of RDF display languages – Language paradigm . . . 80
4.20 Comparison of RDF display languages – Variability, shareability and composability 81
4.21 Fresnel's context-independent presentation knowledge – Selection and ordering . 82
4.22 Fresnel's context-independent presentation knowledge – Formatting . . . 83
4.23 Comparison of RDF display languages – Editor construction & user familiarity . 85
4.24 Comparison of RDF display languages – Resource display, sorting, and formatting 86
4.25 Comparison of RDF display languages – Details of display and formatting abilities 88

5.1 Expressiveness of retinal techniques after Mackinlay . . . 117
5.2 Expressiveness criteria for primitive languages after Mackinlay . . . 117

6.1 Comparison of AVM and SVG . . . 138

7.1 Language levels in the context of RVL . . . 162
7.2 Comparison of techniques to define schemata and derive tooling . . . 164
7.3 Visualisation case coverage – RVL . . . 167

8.1 Process step coverage – Prototypes . . . 190

9.1 Sketches coverage – Implementation and specification . . . 203
9.2 Visualisation case coverage – Prototypes . . . 203

9.3 Requirements coverage – Prototypes . . . 205
B.1 Overview of studied literature . . . 230
D.1 Realised features . . . 237

Listings

4.2 ViZml – Example of a mapping to a colour range 62
4.1 ViZml – Example of a line chart 63
4.3 GPL – Example of a grouped bar graph 63
4.4 Protovis – Example of a bar chart 66
4.5 GSS – Example of a style 72
4.6 REMM – Example of an editing instruction 74
8.1 SPIN – Intra-object constraint (error) 174
8.2 SPIN – Expressiveness constraint (error) 175
8.3 SPIN – Effectiveness constraint (warning and quickfix) 175
8.4 Example of a resource mapping 182
8.5 SPIN – Transformation of resource mappings 182
8.6 Generated property mapping and value mapping 184
8.7 JSON format for processing the AVM with D3 188
8.8 SPIN – Processing an RVL identity mapping 192
D.1 RO-4b – RVL mappings 233
D.2 CIT-5 – RVL mappings 234
D.3 JSON format for processing the AVM with D3 (hierarchical variant) 238