

UNIVERSITY OF MANNHEIM

DOCTORAL THESIS

---

**i<sup>3</sup>MAGE: Incremental, Interactive,  
Inter-Model Mapping Generation**

---

CHRISTOPH PINKEL

*Doctoral Thesis*

*(Inauguraldissertation zur Erlangung des akademischen Grades eines  
Doktors der Naturwissenschaften der Universität Mannheim)*

Data and Web Science Group  
School of Business Informatics and Mathematics

Mannheim, 2016

*Dean (Dekan):*

Prof. Dr. Heinz Jürgen Müller

*Supervisors (Referenten):*

Referent: Prof. Dr. Heiner STUCKENSCHMIDT

Korreferent: Prof. Dr. Carsten BINNIG

*Day of Defense Talk (Tag des Vortrags):*

7th June 2016

*“In order to compose, all you need to do is remember a tune that nobody else has thought of.”*

Robert Schumann

UNIVERSITY of MANNHEIM

# *Abstract*

School of Business Informatics and Mathematics

Doktor der Naturwissenschaften

## **i<sup>3</sup>MAGE: Incremental, Interactive, Inter-Model Mapping Generation**

by Christoph PINKEL

Data integration is a highly important prerequisite for most enterprise data analyses. While hard in general, a particular concern is about human effort for designing a global integration schema, authoring queries against that schema, and creating mappings to connect data sources with the global schema.

Ontology-based data integration (OBDI), which employs ontologies as a target model, reduces the effort for schema design and usage. On the other side, it requires mappings that are particularly difficult to create. Architects who work with OBDI hence need systems to support the process of mapping development. One key type of tooling to support mapping development is automatic or semi-automatic generation of mapping suggestions. While many such tools exist in the wider sphere of data integration, few are built to work in the case of OBDI, where the inter-model gap between relational input schemata and a target ontology has to be bridged. Among those that support OBDI at all, none so far are fully optimized for this specific case by performing a truly inter-model matching while also leveraging distinct but corresponding aspects of both models.

We propose i<sup>3</sup>MAGE, an approach and a system for automatic and semi-automatic generation of mappings in OBDI. The system is built on generic inter-model matching, and it is optimized in various ways for matching relational source schemata to target ontology schemata. To be truly semi-automatic in every respect, i<sup>3</sup>MAGE works both incrementally, building mappings pay-as-you-go, and interactively in exchange with a human user. We introduce a specialized benchmark and evaluate i<sup>3</sup>MAGE against a number of other approaches. In addition, we provide examples, where i<sup>3</sup>MAGE can be deployed in holistic data integration environments.

# *Acknowledgements*

Throughout my work on this thesis, I have been supported and given advice by so many people, that I could not possibly name them all, although I owe each of them my sincere gratitude.

I would like to thank my supervisor Professor Dr. Heiner Stuckenschmidt. Thank you for your advice, and especially for your support of my research topic right from the beginning. I would like to express my special appreciation and thanks to my advisor Professor Dr. Carsten Binnig, you have been a tremendous mentor for me. I would like to thank you for allowing me to grow as a research scientist, even while my time and resources for academic endeavors were constrained. I would also like to thank Dr. Peter Haase for his support to start my research for this thesis while working at fluid Operations, and for his advice on how to align my research with my responsibilities at the company. For extensive feedback and corrections I would like to thank Professor Dr. Wolfgang May.

As usual, parts of this thesis are based on prior publications. I would especially like to thank all of my collaborators in those various publications for discussing my contributions, and for helping me reach the results and gain the insights that have eventually added up to this work.

A special thanks to my family. Words cannot express how grateful I am for your ongoing support over those several last years. Finally, I would also like to thank all of my friends who supported me in writing, and incited me to strive towards my goal.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>Abbreviations</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Support for RDB2RDF Mapping Development . . . . .	5
1.3 Research Questions and Contributions . . . . .	7
1.4 Outline . . . . .	8
<b>2 Background</b>	<b>10</b>
2.1 Automatic Mapping Generation . . . . .	10
2.2 Terminology and Definitions . . . . .	12
2.3 Related Work . . . . .	15
2.3.1 Schema Mapping . . . . .	15
2.3.1.1 Automatic Schema Matching and Mapping . . . . .	16
2.3.1.2 Ontology Alignment . . . . .	19
2.3.2 Incremental and Interactive Approaches . . . . .	20
2.3.3 Inter-Model Mapping . . . . .	24
2.3.4 Mapping Quality and Benchmarking . . . . .	28
<b>3 i<sup>3</sup>MAGE Approach</b>	<b>32</b>
3.1 Problem Statement and Challenges . . . . .	32
3.1.1 Problem Statement . . . . .	32
3.1.2 Challenges . . . . .	33
3.1.2.1 RDB2RDF Challenges Regarding Modeling Primitives . . . . .	34
3.1.2.2 Basic Mapping Pattern Challenges . . . . .	36
3.1.2.3 Advanced Inter-model Mapping Patterns . . . . .	37

3.1.2.4	Approaches to Pattern Recognition Challenge . . . . .	39
3.1.2.5	Challenges in Incremental Matching . . . . .	42
3.2	Matching and Mapping Generation Approach . . . . .	43
3.2.1	Creating Schema Graphs . . . . .	45
3.2.2	Reasoning and Patterns . . . . .	47
3.2.3	Matching . . . . .	48
3.2.4	Fixpoint Computation . . . . .	48
3.2.5	Mapping Generation . . . . .	49
3.2.6	Incremental and Interactive Mapping Generation . . . . .	50
3.2.6.1	Incremental Mapping . . . . .	50
3.2.6.2	Interactive Feedback . . . . .	52
<b>4</b>	<b>i<sup>3</sup>MAGE System</b>	<b>54</b>
4.1	System Overview and Architecture . . . . .	54
4.1.1	System Architecture . . . . .	55
4.1.2	Implementation Notes on Architecture . . . . .	57
4.2	IncGraph . . . . .	58
4.2.1	Running Example . . . . .	58
4.2.2	Modeling Assumptions . . . . .	59
	Ontology Schema $O \in \mathcal{O}$ : . . . . .	59
	Relational Schema $R \in \mathcal{R}$ : . . . . .	59
4.2.3	IncGraph Model Definition . . . . .	60
4.2.4	Basic IncGraph Construction . . . . .	61
4.2.4.1	Relational Schemata (IncGraph( $R$ )) . . . . .	61
	Table Nodes: . . . . .	62
	Attribute Nodes: . . . . .	62
	Datatype Nodes: . . . . .	62
	Reference Nodes: . . . . .	62
4.2.4.2	Ontologies (IncGraph( $O$ )) . . . . .	63
	Class Nodes: . . . . .	63
	Datatype Property Nodes: . . . . .	63
	Datatype Range Nodes: . . . . .	63
	Object Property Nodes: . . . . .	63
4.2.5	Unifying Annotations . . . . .	64
4.2.6	Self References . . . . .	64
4.2.7	Mapping Corresponding RDB2RDF Patterns . . . . .	66
4.2.8	Ontology Meta Knowledge Annotations . . . . .	70
	Unary Axiom Nodes: . . . . .	71
	Binary Axiom Nodes: . . . . .	71
4.2.9	Reasoning on Ontology Annotations . . . . .	72
4.2.9.1	Sub Classes and Super Classes . . . . .	74
	Sub Classes: . . . . .	74
	Super Classes: . . . . .	74
4.2.9.2	Equivalence and Pseudo Equivalence . . . . .	74
4.2.9.3	Deprecated Classes and Properties . . . . .	75
4.2.10	Relational Pattern Annotations . . . . .	75
	Pattern Nodes: . . . . .	76

---

4.2.11	Shortcut Edges . . . . .	77
4.2.12	Annotations from User Queries . . . . .	77
4.2.13	Implementation Notes on IncGraph Construction . . . . .	79
4.3	Matching Background: Similarity Flooding . . . . .	79
4.4	IncMap Matching . . . . .	81
4.4.1	Basic Matching Process . . . . .	81
4.4.2	Extended IncMap Matching: Edge Activation . . . . .	85
4.4.3	Extended IncMap Matching: Iterative User Feedback . . . . .	86
4.4.4	Implementation Notes on Matching . . . . .	87
4.5	Mapping Generation . . . . .	87
4.5.1	Fully Automatic Mapping Selection . . . . .	88
4.5.2	Semi-automatic Mapping Selection . . . . .	89
4.5.3	Implementation Notes on Mapping Generation . . . . .	89
4.6	Additional System Components . . . . .	89
4.6.1	Input Processing . . . . .	89
4.6.2	Feedback System . . . . .	90
4.6.3	Implementation Notes on Additional Components . . . . .	90
<b>5</b>	<b>Benchmark Design and Evaluation</b>	<b>92</b>
5.1	Overview of Benchmark Design . . . . .	92
5.2	Integration Challenges . . . . .	95
5.2.1	Naming Conflicts . . . . .	95
5.2.2	Structural Heterogeneity . . . . .	95
5.2.2.1	Type Conflicts . . . . .	96
5.2.2.2	Key Conflicts . . . . .	97
5.2.2.3	Dependency Conflicts . . . . .	97
5.2.3	Semantic Heterogeneity . . . . .	99
5.3	Analysis of Mapping Approaches . . . . .	99
5.3.1	Differences in Availability and Relevance of Input . . . . .	99
5.3.2	Differences in Mapping Process . . . . .	100
5.4	RODI Benchmark Suite . . . . .	102
5.4.1	Overview . . . . .	102
5.4.2	Data Sources and Scenarios . . . . .	104
5.4.3	Conference Scenarios . . . . .	104
5.4.3.1	Ontologies . . . . .	104
5.4.3.2	Relational Schemata . . . . .	105
5.4.3.3	Scenario Variants . . . . .	107
5.4.3.4	Data . . . . .	108
5.4.3.5	Queries . . . . .	108
5.4.4	Geodata Domain – Mondial Scenarios . . . . .	109
5.4.5	Oil & Gas Domain – NPD FactPages Scenarios . . . . .	109
5.4.6	Extension Scenarios . . . . .	110
5.4.7	Evaluation Criteria – Scoring Function . . . . .	110
5.4.8	System Requirements . . . . .	113
5.5	Framework Implementation . . . . .	114
5.5.1	Architecture of the Benchmarking Suite . . . . .	114
5.5.2	Details on the Evaluation Phase . . . . .	114

---

5.6	Benchmark Results . . . . .	116
5.6.1	Evaluated Systems . . . . .	116
5.6.2	Experimental Setup . . . . .	118
5.6.2.1	Automatic Experiments . . . . .	118
5.6.2.2	Incremental Experiments . . . . .	118
5.6.3	Automatic Experiments: Results . . . . .	119
5.6.3.1	Overall Conference Domain Results . . . . .	119
5.6.3.2	Overall Results for Geodata and Oil & Gas . . . . .	123
5.6.3.3	Drill-down on Selected Challenges . . . . .	125
5.6.4	Incremental Experiments: Results . . . . .	127
5.6.5	Summary of Results . . . . .	128
<b>6</b>	<b>i<sup>3</sup>MAGE Applications</b>	<b>130</b>
6.1	Overview of i <sup>3</sup> MAGE Use Cases and Applications . . . . .	130
6.1.1	Use Cases . . . . .	130
6.1.2	Application Environments . . . . .	131
6.2	i <sup>3</sup> MAGE Support in DataOps Mapping Editor . . . . .	131
6.2.1	DataOps Overview . . . . .	132
6.2.2	i <sup>3</sup> MAGE Application with DataOps . . . . .	134
6.2.2.1	Bootstrapping . . . . .	136
6.2.2.2	Mapping Suggestions . . . . .	136
6.3	i <sup>3</sup> MAGE in Query Driven Setups . . . . .	138
<b>7</b>	<b>Discussion</b>	<b>142</b>
7.1	Summary . . . . .	142
7.2	Discussion of Results . . . . .	143
7.3	Future Work . . . . .	145
	<b>Bibliography</b>	<b>147</b>

# List of Figures

1.1	Traditional architectures for data analyses over multiple sources, with and without data warehousing . . . . .	2
1.2	Data integration architecture for ODBI . . . . .	3
2.1	High-level view of a typical automatic mapping generation process . . . .	10
2.2	High-level view of automatic incremental, interactive RDB $\rightarrow$ RDF mapping	11
3.1	Simple inter-model matching scenario with persons and their addresses (target ontology, and two alternative relational source schemata, including matches) . . . . .	34
3.2	Inter-model matching scenario with papers, reviews and reviewers . . . .	36
3.3	Extended inter-model matching scenario using advanced patterns . . . . .	38
3.4	Simple inter-model matching using an indirect approach . . . . .	40
3.5	Inter-model matching scenario in an indirect matching approach with and without pattern translation through database reverse engineering . . . . .	41
3.6	High-level view of mapping process with i <sup>3</sup> MAGE . . . . .	44
3.7	Relational schema and ontology with corresponding basic IncGraph representations . . . . .	46
3.8	Schema and ontology with advanced IncGraphs . . . . .	47
3.9	Matching IncGraphs (simplified) . . . . .	49
3.10	Incorporation of feedback . . . . .	51
4.1	i <sup>3</sup> MAGE: High-level, end-user's perspective . . . . .	55
4.2	Overall architecture of i <sup>3</sup> MAGE . . . . .	55
4.3	IncMap architecture . . . . .	57
4.4	Running example for IncGraph construction (input target ontology and relational source schema) . . . . .	59
4.5	Basic IncGraph( $R$ ) . . . . .	62
4.6	Basic IncGraph( $O$ ) . . . . .	64
4.7	IncGraph construction with papers and abstracts . . . . .	65
4.8	Correspondence patterns: class-subclass hierarchy . . . . .	67
4.9	Correspondence patterns: properties . . . . .	68
4.10	Full regular IncGraph( $O$ ) . . . . .	72
4.11	Full regular IncGraph( $R$ ) . . . . .	76
4.12	Full IncGraph( $R$ ) with shortcut edges . . . . .	78
4.13	Similarity Flooding . . . . .	80
4.14	Construction of PCG+ (Simplified) . . . . .	83
5.1	RODI benchmark overview . . . . .	94

---

5.2	Overview of RODI benchmark scenarios . . . . .	102
5.3	RODI framework architecture . . . . .	115
5.4	Overview of result scores in default conference scenarios by different systems. Different i <sup>3</sup> MAGE configurations in shades of green. . . . .	120
6.1	DataOps process . . . . .	133
6.2	Configuration/editing steps for DataOps mappings . . . . .	135
6.3	i <sup>3</sup> MAGE suggestions in DataOps . . . . .	137
6.4	Authoring queries in Optique . . . . .	139
6.5	Query based mapping suggestions . . . . .	141

# List of Tables

4.1	IncMap lexical matchers (default in bold print) . . . . .	82
5.1	Detailed list of specific structural mapping challenges. RDB patterns may correspond to some of the “guiding” ontology axioms. Specific difficulties explain particular hurdles in constructing mappings for those cases, which make the combinations unusually challenging. . . . .	98
5.2	Coverage of structural challenges in default benchmark scenarios. Challenges marked with a check are tested throughout the majority of scenarios. ‘Single scenario’ marks challenges that could only be tested in a dedicated scenario. For dependency conflicts, we test only a part of the challenge (misleading axioms), but no restriction violations. . . . .	103
5.3	Basic scenario variants . . . . .	107
5.4	Example results from a query pair asking for author names (e.g., SQL: SELECT name FROM persons WHERE person_type=2; SPARQL: SELECT ?name WHERE ?p a :Author; foaf:name ?name) . . . . .	113
5.5	Overall scores in conference adjusted naming scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	120
5.6	Overall scores in conference restructured scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	121
5.7	Overall scores in conference special case scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	121
5.8	Overall scores in cross-matching scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	122
5.9	Overall scores in geodata scenario (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	123
5.10	Overall scores in oil & gas scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print. . . . .	123
5.11	Score break-down for queries on different match types with adjusted naming conference scenarios. ‘C’ stands for queries on classes, ‘D’ for data properties, ‘O’ for object properties. . . . .	125
5.12	Score break-down for queries that test n:1 matches in restructured conference domain scenarios. 1:1 and n:1 stand for queries involving 1:1 or n:1 mappings among classes and tables, respectively. . . . .	126
5.13	Score break-down for queries that require 1:n class matches on the Oil & Gas atomic tests scenario. . . . .	126
5.14	Impact of incremental mappings: Numbers for IncMap Complete @k human interactions. Bold print numbers mark first reaching the peak. . . . .	127

# Abbreviations

<b>API</b>	<b>A</b> pplication <b>P</b> rogramming <b>I</b> nterface
<b>CWA</b>	<b>C</b> losed <b>W</b> orld <b>A</b> ssumption
<b>DB</b>	<b>D</b> atabase
<b>DBMS</b>	<b>D</b> atabase <b>M</b> anagement <b>S</b> ystem
<b>DDL</b>	<b>D</b> ata <b>D</b> efinition <b>L</b> anguage
<b>DWH</b>	<b>D</b> ata <b>W</b> arehouse
<b>EER</b>	<b>E</b> xtended <b>E</b> ntity- <b>R</b> elationship
<b>ER</b>	<b>E</b> ntity- <b>R</b> elationship
<b>ETL</b>	<b>E</b> xtract, <b>T</b> ransform, <b>L</b> oad
<b>GAV</b>	<b>G</b> lobal <b>A</b> s <b>V</b> iew
<b>GLAV</b>	<b>G</b> lobal and <b>L</b> ocal <b>A</b> s <b>V</b> iew
<b>GUI</b>	<b>G</b> raphical <b>U</b> ser <b>I</b> nterface
<b>HCI</b>	<b>H</b> uman- <b>C</b> omputer <b>I</b> nteraction
<b>LAV</b>	<b>L</b> ocal <b>A</b> s <b>V</b> iew
<b>MCS</b>	<b>M</b> aximal <b>C</b> ommon <b>S</b> ubgraph
<b>OBDA</b>	<b>O</b> ntology <b>B</b> ased <b>D</b> ata <b>A</b> ccess
<b>OBDI</b>	<b>O</b> ntology <b>B</b> ased <b>D</b> ata <b>I</b> ntegration
<b>OLAP</b>	<b>O</b> nline <b>A</b> nalytical <b>P</b> rocessing
<b>OWA</b>	<b>O</b> pen <b>W</b> orld <b>A</b> ssumption
<b>OWL</b>	<b>W</b> eb <b>O</b> ntology <b>L</b> anguage
<b>R2RML</b>	<b>R</b> DB <b>2</b> <b>R</b> DF <b>M</b> apping <b>L</b> anguage
<b>RDBMS</b>	<b>R</b> elational <b>D</b> atabase <b>M</b> anagement <b>S</b> ystem
<b>RDF</b>	<b>R</b> esource <b>D</b> escription <b>F</b> ramework
<b>RDB2RDF</b>	<b>R</b> elational <b>D</b> atabase to <b>R</b> DF
<b>SPARQL</b>	<b>S</b> PARQL <b>P</b> rotocol <b>A</b> nd <b>R</b> DF <b>Q</b> uery <b>L</b> anguage

<b>SPJ</b>	<b>S</b> elect, <b>P</b> roject, <b>J</b> oin
<b>SQL</b>	<b>S</b> tructured <b>Q</b> uery <b>L</b> anguage
<b>TF/IDF</b>	<b>T</b> erm <b>F</b> requency / <b>I</b> nverse <b>D</b> ocument <b>F</b> requency
<b>UI</b>	<b>U</b> ser <b>I</b> nterface
<b>W3C</b>	<b>W</b> orld <b>W</b> ide <b>W</b> eb <b>C</b> onsortium
<b>XSD</b>	<b>X</b> ML <b>S</b> chema <b>D</b> efinition

*To my wife Tatiana*

# Chapter 1

## Introduction

### 1.1 Motivation

Data integration is a big challenge for the industry dealing with enterprise data, but also in many other application domains like life sciences, or the Web. Today, data has not only reached large volumes, but also comes in a variety of formats. Data integration ([1]) increases the utility of data by providing a unified access point to several data sources. It is also a prerequisite to analyze data from disparate sources, e.g., by correlating them and by identifying important patterns [2, 3].

One of the major challenges in data integration tasks is to address the heterogeneity or variety of data.

Traditionally, in data integration scenarios, dedicated information systems are used to run analyses. They are normally backed by large scale data warehouse systems (DWHs) or, more recently, by big data frameworks such as Hadoop YARN [4] that work as *data operating systems* running a mix of data warehousing and other data-processing applications. In all of these cases, data are typically imported from their original locations using an extract-transform-load process (ETL), which leads to a unified, integrated, global view.

Data warehouse solutions, however, come with a significant downside: they require a dedicated global warehousing schema, which first needs to be carefully designed, and mappings need to be constructed. The resulting schema then is still on a technical, non-conceptual level, and consumers need to interact with the integrated data on that level. This comes at the price of either a limited set of globally accessible data and query support with little flexibility. Or, if the scope of access and flexibility needs to increase, it requires even more effort for programming all the tasks and queries to be supported.

Worse, with a number of data sources that quickly change in structure, maintenance of the integrated global schema, mappings and queries can quickly become very difficult.

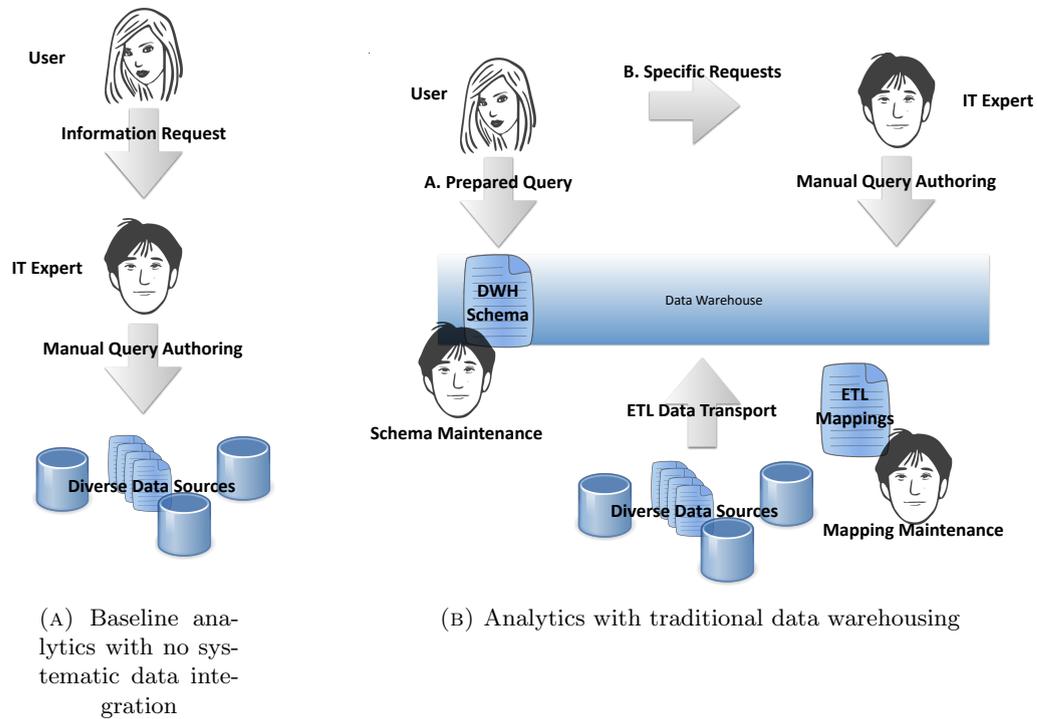


FIGURE 1.1: Traditional architectures for data analyses over multiple sources, with and without data warehousing

Figure 1.1 depicts the exemplary schematics of integrative data analytics without any specific tooling at all (Figure 1.1a) and with a traditional data warehouse architecture (Figure 1.1b). If no specific data integration systems are employed at all, any information requests that span several data sources need to be “programmed” by one or several IT experts. This works well for a very small number of queries that do not change very often, but causes intense effort with a larger, changing or growing number of diverse information requests. A data warehouse allows users to pose queries against a unified global schema, but shifts the effort into designing and maintaining that schema as well as the mappings that translate from the original data sources. Also, because the global schema still remains technical in nature, end users typically still rely on the assistance of IT experts for all but a limited number of prepared standard queries.

For scenarios with a high degree of schema complexity in the data sources or with a large number of data sources with only partially overlapping schemata, the problem of upfront setup effort becomes particularly pressing. A global schema reflects all of the data sources considered in the scenario. Its size and structural complexity therefore approximately reflect the sum of the sizes and complex structural elements from all non-overlapping parts of all data sources involved.

Often enough, the effort for setup and maintenance becomes unacceptable. This contributes to the current situation, where enterprises are assumed to analyze less than one sixth of their potentially relevant data.<sup>1</sup>

A promising recent approach to address part of this predicament is to use ontologies, semantically rich conceptual models [6], to provide a conceptual integration and an access layer on top of data sources [7]. Ontologies then act as the global, integrated schema. This approach is referred to as either *ontology-based data access* (OBDA) when applied to a single data source, or as *ontology-based data integration* (OBDI) in the more general case. The approach has been successfully applied in academia as well as in the industry [8–11]. Ontologies are connected to databases with the help of mappings that describe the relationship between the elements of relational database schemata and the ontological vocabulary, expressed in RDF. They are therefore referred to as RDB2RDF (Relational Database to RDF) mappings.

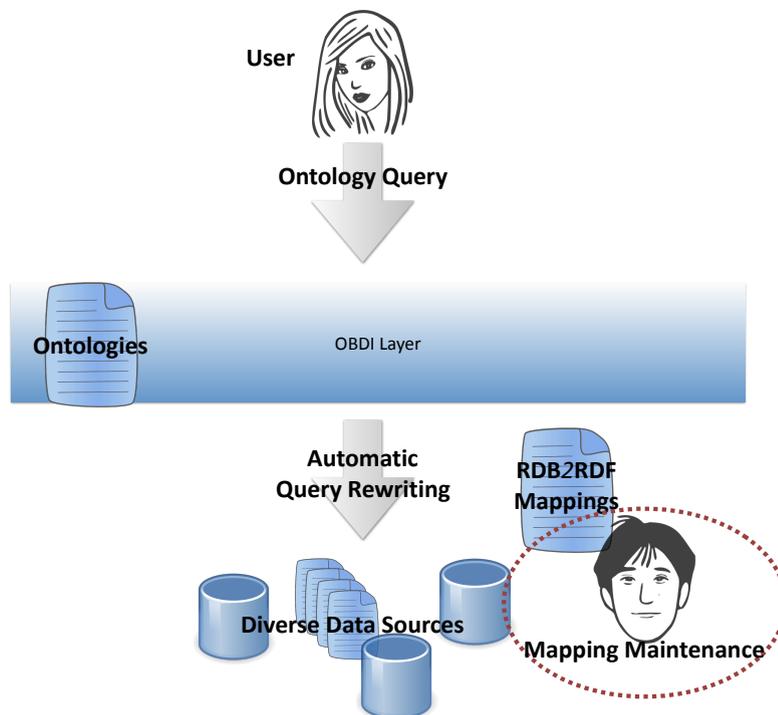


FIGURE 1.2: Data integration architecture for OBDI

Figure 1.2 schematically depicts an OBDI data integration system.

Other than in traditional data warehousing, the ontologies used in OBDI mediate between the data and its consumers. Consequently, users (or applications) who wish to access the data can formulate semantically rich queries in their own high-level view of a conceptual domain model represented by the ontology. This enables them to phrase

<sup>1</sup>According to a recent study of business analysts [5], surveying several hundreds of enterprises.

a much wider range of information needs directly against the system and without the intervention of an IT expert. It also means, that an ontology that acts as a global schema reflects more closely the conceptual properties of the application domain and is therefore less dependent on technical properties of the source schemata. Other than traditional integration schemata they thus do not normally need to be changed along with structural changes in data sources.

Also, expert ontologies are already publicly available in many application domains, and many of them can naturally be employed to support OBDI scenarios. For example, in biology there is the Gene Ontology [12], and in medicine [13] there is the International Classification of Diseases (ICD) ontology. Another recent example is [schema.org](https://schema.org),<sup>2</sup> an ontology to mark up data on the web with schema information. Industrial examples include the NPD FactPages ontology [14] created for oil exploration and the Siemens ontology [10] for turbine hardware and sensor measurements.

In many cases with OBDI it is therefore not necessary to design a new global schema from scratch.

Still, OBDI crucially depends on the quality of not only ontologies but also on RDB2RDF mappings that connect relational databases to RDF-based ontologies.

Mapping development has however been given much less attention than the development of ontologies has seen. Moreover, existing mappings are typically tailored to relate generic ontologies to *specific* database schemata. As the result, in contrast to ontologies, mappings typically cannot be reused across integration scenarios. Thus, each new OBDI scenario essentially still requires the development of mappings from scratch.

Figure 1.2 highlights RDB2RDF mapping development and maintenance as the major remaining bottleneck in OBDI. In fact, mapping development for OBDI could be considered an even more difficult task than mapping development in general. This is because with RDB2RDF mappings, the cognitive accomplishment of translating from underlying, low-level data source models to a conceptual high-level target ontology has to be encoded in the mappings. Both of the other labor-intensive tasks in data warehousing that normally require IT experts, i.e., schema maintenance and the formulation of individual queries, have been mostly alleviated in OBDI. Mapping development still remains a complex and time-consuming process.

---

<sup>2</sup><https://schema.org>

## 1.2 Support for RDB2RDF Mapping Development

As mapping development requires significant time in OBDI, support for users needs to be addressed in order to achieve the overarching aim of reducing the human effort in complex data integration.

More broadly, mapping creation and maintenance are a part of the issues that make data integration hard in general.

For instance, Doan et al. [1] take an introductory, high-level view on hard challenges surrounding data integration. They group those challenges into four categories: they start with *systems reasons* and *logical reasons*, which both are technical. Then, they discuss *social and administrative reasons* as well as *setting expectations* as the remaining two categories, which are motivational. On the motivational side, i.e., expectations, the authors of [1] also make clear that the first aim is to reduce human effort, with accuracy often being a competing goal.

Assuming this perspective, support could be provided from different angles. Chiefly, users could be assisted with technical challenges, e.g., by automatically proposing mappings, by visualizing complex logical connections between data sources, or by providing a suitable user interface to translate the mapping idea of a user into a formal language. They could also be supported with managing the motivational challenges, e.g., by means of matching their expectations with available assets, or by guiding them through the mapping creation process along the lines of administrative requirements. It should be the aim of any technical system in the field of data integration to *solve* some parts of the technical problems, but at the same time appropriately *consider* the soft factors. In particular, any supportive approach should be designed with the consideration that other forms of support may be eventually needed in addition to and in combination with it.

One key approach to offer support in developing and maintaining mappings is automatic or semi-automatic support for mapping generation. Systems that help constructing mappings of good quality are therefore needed.

Numerous such systems exist in the wider sphere of data integration. Because the approaches need to scale with the size and complexity of schemata and as fully automatic solutions are usually insufficient in terms of quality, it has been widely accepted that such tools should be semi-automatic systems (c.f., [15–17]) and highly specialized for the task.

Consequently, a number of systems have also recently been developed to address mapping generation support for the special case of RDB2RDF [18–22]. In addition, a few earlier systems also support some flavor of RDB2RDF mapping generation [23, 24].

All of those systems, however, fail to fully address the particular issues of the inter-model gap that make RDB2RDF special. For instance, as relational source databases yield no explicitly modeled semantics, it cannot be automatically expected, that general-purpose reasoning works as effectively as is known from ontology alignment [25, 26]. General-purpose matching algorithms, like graph-based structural matching, therefore would appear more promising at first. Still, the target schema is an ontology, and also source relational schemata contain at least some implicit semantics. It is therefore still reasonable to assume that reasoning techniques can also have a positive impact if and when tuned and applied to the specific situation. Additionally, in any inter-model gap there potentially exist corresponding pairs of typical design patterns, which may look largely different in either model but are frequently used to model the same type of information.

We therefore argue that a specialized system, which considers the specific particularities of RDB2RDF inter-model mapping generation, should hypothetically be capable of producing better results.

Although some approaches take steps into the direction, a system that systematically considers the RDB2RDF inter-model gap has not yet been described in the literature. A prominent early system, COMA++ [24], has addressed the problem of inter-model gaps by generalizing all kind of models to a common structural graph. This makes inter-model matchings generally feasible between any two supported models. The system does not, however, consider the particularities of those different input models. It does also not provide any particular optimizations for inter-model matching. For instance, correspondences between design patterns in RDB2RDF are not considered. More specialized early approaches like RONTTO [23] consider what we now refer to as basic inter-model patterns. RONTTO does not combine this approach with generic inter-model matching techniques such as the ones employed by COMA++, though. Recent systems like BootOX [19] combine ontology alignment with some consideration of the particularities of relational source schemata. Again, BootOX does not consider correspondence patterns or generic structural properties.

Our initial field experience with semi-automatic mapping generation support has also lead us to the assumption that specialized optimizations for RDB2RDF may be necessary to bridge the inter-model gap effectively.

We therefore propose a system, which addresses the specific problems of interactive mapping generation in RDB2RDF inter-model settings. The system should aim to reduce the overall human effort in the process of creating *sufficiently accurate* mappings. Sufficient accuracy, of course, is eventually laid out by user requirements and in the scope of our motivational scenario usually refers to perfect accuracy w.r.t. a certain set of tasks or queries. On the technical side, we therefore first aim to generate mapping suggestions that are as close to the eventually expected mapping as possible, optimizing on particular technical challenges of the inter-model use case. However, such a system should also attempt to provide suggestions in a way that makes them easy to process in user interactions, allows them to improve incrementally on user feedback, and fits into a wider mapping development process where other forms of support could be leveraged simultaneously.

### 1.3 Research Questions and Contributions

This work aims to tackle the specific issues in incremental, interactive schema mapping generation in the case of RDB2RDF inter-model mappings. We build on established best practices in schema matching and focus on unsolved issues that occur exclusively or primarily in the context of inter-model matching, while all contributions are designed to work in an incremental and interactive setup. In particular, we address the research questions listed below and make contributions as follows:

- *What are the specific challenges of inter-model mapping generation as opposed to generating regular intra-model mappings, specifically w.r.t. RDB2RDF mappings?*

We discuss specific challenges and previous approaches that attempt to address those challenges and point out gaps and shortcomings that have not been sufficiently addressed to date. To this end we bring together observations of design patterns from the fields of traditional (relational) database management, schema matching and ontology matching and discuss their joint impact on RDB2RDF inter-model matching. We also analyze how partial mappings of complex schemata can be exploited in such a setting to improve match suggestion quality incrementally.

- *How can mapping generation systems be designed to provide enhanced support for those specific RDB2RDF mapping generation challenges?*

We discuss potential specific features and introduce i<sup>3</sup>MAGE, a system that combines several such features for automated RDB2RDF mapping generation. i<sup>3</sup>MAGE

uses a combination of lexical, structural and logical features in ontologies and relational database schemata to generate tailored suggestions. Besides specific adjustments to established matching best practices, we propose new measures to optimize mapping generation for RDB2RDF, including the consideration of systematic correspondences between design patterns.

- *How can the quality of generated RDB2RDF mappings be measured w.r.t. real-world utility and how do specialized approaches compare to the state of the art?*

We discuss the requirements for a broadly applicable quality benchmark of automatically generated RDB2RDF mappings. As no such benchmark has been previously available for RDB2RDF mapping generation, we design a benchmark according to those requirements. We provide a broad experimental evaluation of i<sup>3</sup>MAGE and several other systems in the field to identify their performance, strengths and weaknesses. Beside our own evaluation, the benchmark has since also been applied by several research groups to test their own approaches.

- *How can user-feedback and other context be exploited to gradually improve the quality of generated mappings?*

We discuss prerequisites and opportunities to make i<sup>3</sup>MAGE semi-automatic along two related dimensions, incremental, pay-as-you-go development of mappings, and interactive user feedback. As a result, we implement and evaluate the effect of iterative user feedback to refine mapping suggestions, but also process partial mappings and query workloads as additional context.

- *How can such generated mappings be integrated non-intrusively in a semi-automatic process?*

We present use cases and system environments that may make use of i<sup>3</sup>MAGE, and discuss requirements to effectively include i<sup>3</sup>MAGE in these environments. Additionally, we present prototypical implementations of i<sup>3</sup>MAGE in two such environments, demonstrating how i<sup>3</sup>MAGE can be used in real-world end-to-end use cases.

## 1.4 Outline

The remainder of this work is structured as follows: We first discuss the background on automatic and semi-automatic matching and mapping generation in Chapter 2, and give an overview of related work.

In Chapter 3 we introduce our i<sup>3</sup>MAGE approach to RDB2RDF mapping generation, including the high-level idea and rationale for inter-model graph matching, our take on

---

RDB2RDF optimizations, and our approach to support incremental and interactive processes. Then, Chapter 4 presents the technical implementation of i<sup>3</sup>MAGE, including a formal definition of the graph matching model and optimizations, as well as implementation details.

Chapter 5 introduces the RODI benchmark suite that we have designed to evaluate the quality of automatically generated RDB2RDF mappings and gives a detailed comparison of i<sup>3</sup>MAGE vis-a-vis other systems in the field. Finally, we discuss application scenarios, where i<sup>3</sup>MAGE has been deployed in Chapter 6, and we exemplify how the different system capabilities could be used in practice.

We conclude in Chapter 7 by summarizing and discussing our findings and contributions and by pointing to possible avenues of future work.

## Chapter 2

# Background

In this chapter, we start with an introduction to the high-level idea of automatic mapping generation in Section 2.1. We introduce basic terminology that we will use throughout the remainder of this work and give definitions of the key terms in Section 2.2. We then discuss the state of the art in related literature in Section 2.3.

### 2.1 Automatic Mapping Generation

Simply speaking, mappings mediate or translate between two or more non-compatible sets of data.

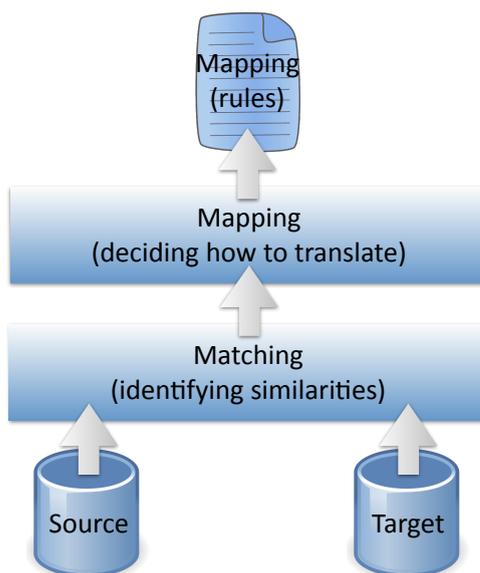


FIGURE 2.1: High-level view of a typical automatic mapping generation process

Figure 2.1 depicts a typical process for automatic mapping generation in the general case. As input, a system takes the two (or more) different data sets and eventually outputs a mapping. The process is often implemented in two steps: during *matching*, similarities between the different input data sets are being defined. For instance, matching could determine that some  $A$  corresponds to some  $B$  with a similarity score of 0.61, but  $A$  also corresponds to some  $C$  with a score of 0.85. Then, during *mapping*, the system acts on those identified similarities to produce mappings in some sort of (often formal) mapping language. However, those two aspects may not necessarily take the shape of clearly separated steps in implementation.

This work concerns itself with a novel approach and a system for automatic incremental, inter-model mapping generation with interactive user feedback. More specifically, the approach is focused on the particular case of schema mapping between relational databases and RDF-based ontologies (RDB $\rightarrow$ RDF). Thus, in our case, the input to automatic mapping generation will mostly be a relational schema and an ontology. Also, inputs and outputs may be incomplete at various stages for incremental mapping generation, and partial mappings may become input to future iterations. Similarly, user feedback concerning previous results and other user input needs to be taken into consideration.

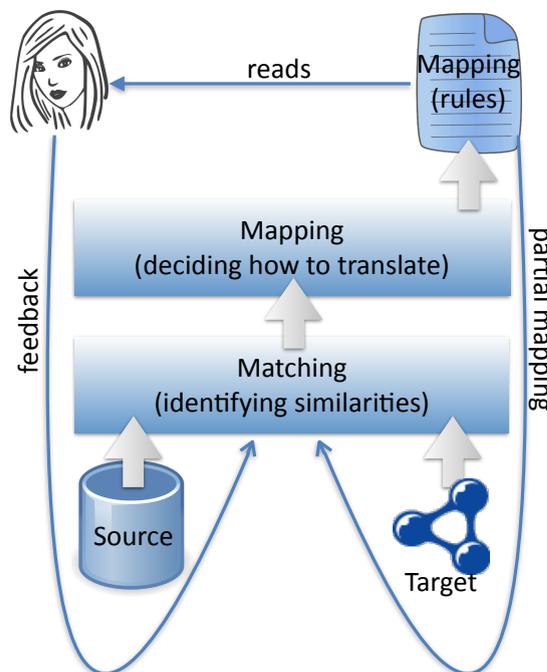


FIGURE 2.2: High-level view of automatic incremental, interactive RDB $\rightarrow$ RDF mapping

Still largely simplifying the process, Figure 2.2 depicts a prototypical architecture for the automatic generation of such incremental, interactive, inter-model mappings as we have

them in our case. Here, users can be involved, and both their feedback and previous mappings can later be used as subsequent input.

## 2.2 Terminology and Definitions

We tackle problems in the scope of automatic schema mapping generation, more specifically for the particular case of semi-automatic RDB2RDF schema mapping generation.

**Definition 2.1** (Modeling Languages, Schema Elements). We refer to languages that are designed to model data according to some structure as *modeling languages*. We denote the set of all such modeling languages as  $\mathcal{L}$ . For each language  $L \in \mathcal{L}$ , we denote modeling primitives  $p \in L$  as *schema elements*.

Examples of modeling languages include the relational model as well as different ontology languages. Schema elements comprise aspects such as, e.g., class definitions in the case of an ontology language.

**Definition 2.2** (Schemata and Instances). For any schema  $S$  in language  $L \in \mathcal{L}$ , we call  $D$  a *database* of  $S$  or an *instance* of  $S$ , if  $D$  adheres to the model described by  $S$ . We use  $\mathcal{S}$  to denote the set of all schemata, independent of the language in which they are expressed, and  $\mathcal{D}$  for the set of all databases. By *instanceOf*( $D, S$ ), we denote that a database  $D \in \mathcal{D}$  is an instance of a specific schema  $S \in \mathcal{S}$ .

In case of ontologies, a schema is typically given by means of the ontology's T-Box, with axioms as schema elements.<sup>1</sup> For relational databases, the schema is a relational schema in a modeling language based on what has been originally described as the relational model [27]. In practice, relational schemata are today expressed in DDL (Data Definition Language, c.f. [28, Chap. 2.3]).

More specifically:

**Definition 2.3** (Ontology Schema). For schemata  $O$  that are modeled in an ontology language, we call  $O \in \mathcal{O} \subset \mathcal{S}$  an ontology schema.

Instances of ontology schemata usually coincide with the A-Box of the ontology.

We leave the exact definition of ontology languages open, as different but equally legitimate definitions could be applied for different cases of schema mapping generation. In practice, we will mostly refer to ontology languages in terms of the Web Ontology Language (OWL) [29].

---

<sup>1</sup>Depending on the expressivity of the T-Box, not all of its axioms might be considered as relevant parts of the schema.

**Definition 2.4** (Relational Schema). We call a schema  $R$  a relational schema, if it is modeled as a constrained enumeration of relation signatures. We use  $\mathcal{R} \subset \mathcal{S}$  to denote the set of all relational schemata. In this work we represent relational schemata as a set of attribute definitions, relation definitions and constraints, i.e.,  $R \subset \{r_1, r_2, \dots\}, \forall r_i : relation(r_i) \vee attribute(r_i) \vee constraint(r_i)$ .

Instances of relational schemata are any collections of attribute tuples that fit into the relations defined by the schema and do not violate any constraints of that schema.

Mappings between any two schemata are described by mapping rules:

**Definition 2.5** (Schema Mapping Rules). Let  $S, T \in \mathcal{S}$  be a source schema and target schema, respectively. Then mapping rules are a set of rules  $M \in \mathcal{M}$  to rewrite data from instances of  $S$  to instances of  $T$ :  $\forall D_S \in \mathcal{D} \wedge instanceOf(D_S, S) : \text{there is a transformation function } t$ , which can produce a target database from a source database using these mapping rules, i.e.,  $\exists t : (\mathcal{D}, \mathcal{M}) \rightarrow \mathcal{D}, t(D_S, M) = D_T \wedge instanceOf(D_T, T)$

In the following, we refer to schema mapping rules simply as mapping rules.

Note, that our definitions so far mostly follow the definition of *semantic GLAV* (Global and Local as View) mappings according to [1, Chap. 3]. We leave the semantics of transformation rules (i.e., query semantics) more open, though. We also do not require a logically sound transformation of one schema into another, but rather require only their contents (i.e., any data instances) to be translatable. This is in contrast to some stricter definitions that often require a formally defined mapping language. In general, however, there is a large degree of divergence in definitions. Ten Cate et al. [30] summarize schema mapping definitions as “high-level specifications that describe the relationship between two [schemata] (...) typically expressed in declarative languages based on logical formalisms”. This definition largely fits our needs in this work, although we have no strict requirement for any of the aspects described as “typical” in this definition: both the use of a declarative language and a foundation in logical formalisms are optional according to our own definition. Also, a relation between two schemata, in our case, is sufficiently specified if it can be applied to somehow rewrite a database that is described by one schema into a database described by the other. Through this relaxation compared to some other definitions we gain the flexibility to apply complex inter-model mappings in a pragmatic and useful way, even without necessarily having full knowledge of formal and universal translation semantics between the models. On the other side, we lose the ability to formally prove equality or subsumption of the schemata under given mappings and thus will require different quality measures.

**Definition 2.6** (Mapping Instances). For source schema  $S \in \mathcal{S}$  and target schema  $T \in \mathcal{S}$ , let  $M \in \mathcal{M}$  be a set of mapping rules between  $S$  and  $T$ . Then, a mapping instance  $I \in \mathcal{I}$  can be defined as a tuple  $I = (S, T, M)$ .

When producing mappings, the initial state of a mapping instance is usually still lacking any mapping rules, i.e.,  $I_{init} = (S, T, \{\})$ . Mapping rules can then be added either at once or during the course of several iterations that continuously refine the mapping instance, i.e.,  $I_1 = (S, T, M_1), I_2 = (S, T, M_2), \dots$ . Changing one mapping instance into another can be a manual rewriting step or could be automated.

**Definition 2.7** (Mapping Generator). A mapping generator is a function  $g$  that rewrites a mapping instance into another, changing only mapping rules. That is,  $g : \mathcal{I} \rightarrow \mathcal{I}, g(S, T, M) = (S, T, M')$

Note, that some broader definitions have been used in the literature as well (e.g., [31]). In those definitions, modifications to the target schema are allowed or required during mapping generation (e.g.,  $g_{relaxed} : \mathcal{I} \rightarrow \mathcal{I}, g_2(S, T, M) = (S, T', M')$ ). We focus on the strict case where a mapping generator produces *only* mappings, i.e., mappings between two given sides, source and target.

A mapping generator may take into account additional context, if available. Such additional context could, for instance, be a sample database or a query workload on either  $S$  or  $T$ , or human input. Formally, you may think of such a generator as a higher-order function, which takes context as input and yields a final mapping generator function.

In this work we consider mapping generation in incremental, interactive scenarios. That is, we consider scenarios where several mapping generation functions can be chained, and additional context becomes available in-between each step, following human interaction.

**Definition 2.8** (Mapping Task). A mapping task is the task to transform any mapping instance  $I$  to bring it closer to meeting user-specified expectations. Formally, a mapping task is defined by a mapping instance and a utility function or a measure of success, i.e., a pair  $(I \in \mathcal{I}, u : \mathcal{I} \rightarrow [0..1])$ .

A mapping generator can be judged by the gain in utility that results from its rewriting of a mapping instance.

For instance, a simple mapping task in a testing or benchmark setting could involve a source schema  $S$ , a target schema  $T$  an initial mapping instance  $I_{init} = (S, T, \{\})$ , and a reference mapping  $M_{ref}$  with reference mapping instance  $I_{ref} = (S, T, M_{ref})$ . In that case, the mapping task would be to rewrite  $I_{init}$  into some  $I_{result}$  by adding mapping

rules in such a way, that  $I_{result}$  closely resembles  $I_{ref}$ . A utility function could measure success by executing both  $I_{result}$  and  $I_{ref}$  and compare the resulting databases in terms of precision and recall.

**Definition 2.9** (Multi-source Integration). Multi-source integration following this definition is set of mapping tasks, sharing the same target schema:  $I = \{(S_1, T, M_1), (S_2, T, M_2), \dots\}$ .

In this work, we mainly consider individual mapping tasks w.r.t. mapping generation, although some cases of multi-source schema integration are being discussed.

We primarily aim to tackle mapping tasks for RDB2RDF, i.e., to generate mappings with the goal of optimizing mapping utility in a relational-to-ontology inter-model setting, possibly over several iterations.

## 2.3 Related Work

We start by summarizing the state of the art in general schema mapping, including automatic approaches in Section 2.3.1. Then, we discuss existing incremental and interactive approaches in Section 2.3.2 and previous work on inter-model mapping in Section 2.3.3. Finally, in Section 2.3.4 we give an overview of previous approaches to evaluate the quality of automatically generated mappings, in particular for RDB2RDF, and discuss related work on benchmarks.

### 2.3.1 Schema Mapping

Schema mapping is a key aspect of data integration, which is a well studied research problem [1, 3]. As such, the field of schema mapping has seen research efforts as far back as the early 1980s (e.g., [32]). Roughly ten years on, the field gained momentum with new architecture proposals (e.g., [33]). Later, it became a topic of even more intense research, in particular in the context of enterprise information integration (e.g., [34–36]).

An early survey on schema integration methodologies has been provided by Batini et al. [37]. Later surveys include the ones of Ouksel and Sheth [38], or by Doan and Halevy [39]. As most later surveys in the increasingly broadening field, however, those papers assume a rather selective and focused point of view. Data integration textbooks (e.g., [1]) give a more general overview of methodologies, approaches, and technologies, as well as of the history of schema integration.

Mapping rules can be formulated in various ways. A common definition considers mappings as declarative constraining relations between source and target (Miller et al. [40]). Our definition instead follows the observation that, in enterprise practice, mappings cannot always easily be expressed or understood in basic declarative terms. We thus adopt the more relaxed definition where mapping rules are (possibly Turing complete) procedural transformation rules. This is the kind of mappings typically used in ETL-based data integration systems [1, Chap. 1].<sup>2</sup> For the more formal approach, a raft of logical mapping languages has been proposed. Detailed comparisons between the different mapping language formalisms have been made by Ullman [41], Lenzerini [42], or ten Cate and Kolaitis [30].

### 2.3.1.1 Automatic Schema Matching and Mapping

Automatic mapping is the approach to solving (or partially solving) mapping tasks automatically using a mapping generator. It thus constitutes a fundamental aspect at the heart of i<sup>3</sup>MAGE. The field of automatic mapping is generally broad and not bound to the exact scope of mapping tasks as per our previous Definition 2.8, i.e., the task to (re-)write mapping rules with the aim of improving w.r.t. some utility measure.

Basically all approaches to automatic mapping, including our own, use a pool of common fundamental *matching techniques* [43], though. The most common ones of those can be roughly grouped into three categories:

1. Lexical matching, essentially based on string matching and similar techniques [44], often from the field of Information Retrieval. Elements in a schema are considered similar, if their labels or descriptions are similar.
2. Structural matching, based on the similarity of either fixed structural patterns or using graph matching (e.g., [45]). Elements in a schema are then considered similar, if they are embedded in a similar or comparable structure. A special case of structural matching are logical rules, which could be used to either define and detect structural patterns or to even reason on the consequences of explicitly known semantics.
3. Third, matching with auxiliary information such as thesauri or dictionaries of various kinds. Elements in a schema are considered similar, if available auxiliary information indicates that they are described in terms that are semantically similar (e.g. [46]).

---

<sup>2</sup>i<sup>3</sup>MAGE is not an ETL system and mappings produced by i<sup>3</sup>MAGE are declarative. However, i<sup>3</sup>MAGE could be used in ETL context and we intend to keep it comparable with systems that employ non-declarative mappings.

Matches can be considered building blocks to automatic mapping, where a match describes the degree of similarity between any two structural elements of the schemata on either side. Matches are also often interchangeably referred to as *correspondences* [1, Chap. 5].

In the context of schema mapping, automatic mapping is even sometimes referred to as schema matching. Although slightly inaccurate, this pays tribute to the fact that generated matches are the foundation for generated mappings, and in some cases matching and mapping are inseparable. The COMA++ paper [24] is an example where authors do not differentiate between match results and mappings, instead saying that “the obtained mapping is a set of correspondences” [24, p. 907].

A general overview of different matching techniques can be found in the surveys of Shvaiko and Euzenat [47] or in the more recent survey of Bernstein et al. [48].

A significant number of schema matching and mapping systems have been described in the literature, including Clio [17], Cupid [49], Artemis [46], MOMIS [50], Similarity Flooding [45], COMA [16], and AgreementMaker [51].

Of those, Similarity Flooding is most closely related to our approach, as our core component IncMap [52, 53] uses the matching principles of Similarity Flooding. In general, Clio and COMA take a particularly prominent position as a point of reference for many later systems.

Similarity Flooding [45, 54] is rather a generic algorithm with associated data structures than a complete system, although the authors have implemented a complete prototype for experimental purposes. The approach builds around graph matching in a fixpoint computation algorithm. In [45], the authors therefore devise two general purpose graph structures that can be used in the algorithm, a *pairwise connectivity graph* and an *induced similarity propagation graph*. A more detailed discussion of both data structures can be found in [54]. As a prerequisite to applying Similarity Flooding, compatible graph representations of the source and target schema with colored nodes and generic, labeled edges have to be produced. For their own experiments, the authors devise such graphs in particular for relational databases and in a most straightforward manner. In IncMap we use IncGraph, which is a compatible input to Similarity Flooding. The pairwise connectivity graph then results from a Cartesian product of the nodes of input graphs of the same color. Nodes in the graph need initial similarities, which can be assigned using arbitrary lexical matching methods. From the pairwise connectivity graph, the induced similarity propagation graph can be constructed. Slightly simplifying, induced similarity graphs can be considered as another representation of the pairwise connectivity graph, with changes in edges and weights to fit the following fixpoint computation.

Intuitively, the idea behind the fixpoint computation is to favor nodes in larger sub-graphs over smaller ones and to increase the scores of strongly connected nodes. The fixpoint computation runs in several iterations with score normalization between each two iterations. It ends after either no more significant changes to scores happen, or after reaching a configurable maximum number of iterations. We discuss all aspects of Similarity Flooding that are used in IncMap in greater detail in Chapter 4.

Clio [17, 55, 56] was one of the earliest advanced research prototypes in automatic schema matching, although it is described in limited detail in the literature. It has also later made its way into commercial mapping technologies at IBM [55]. While originally developed for matching relational schemata, the system has later been extended to also support XML. In contrast to most other systems, Clio puts the actual *mapping* generation at the center of its efforts and considers *matches* merely as an input produced by a schema matching component, which is not discussed in detail. It therefore works with simple matches and follows a tableaux-based approach to calculate complex mappings on that basis. One of its strong suits is its RDB-specific advanced support for relational constraints and join paths. Clio supports an internal logical representation of mappings and can export them in several query languages for actual transformations.

COMA [16] builds on two core ideas: combining different match algorithms in a flexible way and enabling the reuse of results from previous match operations. At the core of the system, a library of matchers supports different matching strategies by choosing any matcher or a combination of several matchers. The library is extensible not only with future increases of functionality in mind but also to enable the use of COMA as a framework for evaluating matchers or their combinations. Matchers are meant to be combined in a *composite* approach, aggregating their different scores for match candidates to derive a combined similarity value. All matchers work on a common graph structure representation of the input schemata, which supports different models, such as relational schemata or XML. The graph represents *schema elements* in a similar fashion to IncGraph, although it uses different edge semantics and does not use the graph structure to model any additional knowledge, patterns or other heuristics. Also, schema elements in COMA represent paths of arbitrary length rather than individual nodes. While lending less weight to overall structural features within the graph, this naturally accommodates matches between any two paths of different lengths. COMA's own evaluation demonstrates high accuracy w.r.t. expectations set by the authors and also some practical usefulness on a selection of medium-sized schemata. COMA has later been improved and republished in an extended version as COMA++ [24]. Where the original version did already feature generic, model-independent data representation with a focus on relational schemata and XML, COMA++ also supports OWL. Inter-model matchings between any two different data models are in principle supported.

However, while matchers can cater to the characteristics of different models individually, no dedicated inter-model matchers are available. Consequently, the tool is not assisting the user in bridging the impedance mismatch, and it could be assumed that it would perform better in intra-model matchings than it does in complex inter-model matchings.

Besides published works, a series of commercial systems has emerged as well. Among the most prominent are Altova's MapForce<sup>3</sup>, IBM's Rational Data Architect<sup>4</sup> and Microsoft's BizTalk.<sup>5</sup>

### 2.3.1.2 Ontology Alignment

Schema matching between ontologies is either referred to as ontology matching or as *ontology alignment*. To some degree this may simply be a result of the fact that ontology alignment has been investigated largely independently from other schema matching research in its own community. However, schema matching between ontologies differs significantly from other types of schema matching, and these differences also warrant for a term on its own.

While ontology matchers often employ the same basic matching techniques as regular schema matchers, they can also apply logical reasoning. This is because, in contrast to all other popular schema models, ontologies as a schema contain explicit and often rich semantics. For the same reason, there is usually even less a distinction between matching and mapping in ontology alignment than there is in schema matching and mapping: logical correspondences between schema elements with explicit semantics (e.g., class equivalence or entailment) are already sufficient to deterministically derive a complete mapping. Hence, there is no room for research on appropriate mapping heuristics or mapping algorithms drawing on additional external information.

Ontology aligners include PROMPT [57], LogMap [58], CODI [59], AIViz [60], SAMBO [61], RiMOM [62], YAM++ [63], and others.

Further actively developed systems can be found in the yearly lists of participants and results of the OAEI [25] (Ontology Alignment Evaluation Initiative), e.g., [26].

From the family of traditional schema matchers discussed in the previous Section 2.3.1.1, COMA++ [24] and AgreementMaker [51] also additionally qualify as ontology aligners.

For surveys, see [64] or [65].

---

<sup>3</sup><http://www.altova.com>

<sup>4</sup>Recently rebranded as InfoSphere Data Architect, <http://www.ibm.com/software/products/en/ibminfodataarch>

<sup>5</sup><http://www.microsoft.com/en-us/server-cloud/products/biztalk/>

### 2.3.2 Incremental and Interactive Approaches

Quite generally, it has been a long accepted fact that purely automatic mapping approaches will often fall foul of quality expectations (c.f. [1, 15, 16, 66]). A lot of research efforts have therefore been made in the field of semi-automatic mapping approaches.

Examples in traditional schema matching include most established systems (e.g., Clio [17]) as well as basically all commercially established systems.

In ontology alignment, examples include PROMPT [57], AlViz [60], NeOn's [67] On-toMap and others. For RDB2RDF, RONTA [23] is an early example, and BootOX [19, 68] a more recent one. While being semi-automatic and thus interactive, none of them follows a truly incremental approach. Instead, the more traditional approach is the one described in detail by Clio [55], which runs in three stages. First, fully automatic matching is performed. Then, a human user is being confronted with all of those matches, and can add, remove or change matches. Finally, mappings are generated based on the corrected matches. At best, steps one and two can be repeated several times, leading up to an iterative process. Consequently, they offer poor support for constructing a mapping part by part, subsequent automatic mappings cannot significantly leverage on feedback or partial mappings from earlier steps.

Approaches that do involve user interaction for incremental mapping generation include [69–72]. Typically, these pay-as-you-go approaches assume a classical mass scenario with a large number of users, massive but simple *end-user* feedback, a lot of noise and *statistical* methods to harvest feedback. This is in contrast to our approach of requesting very explicit feedback from a small number of *expert* users. Similarly, classical human computer interaction and, more recently, crowd sourcing have been investigated (e.g., [73]) but they remain just as limited in perspective of seeing users as a large sample to be observed statistically.

Karma [18] is semi-automatic in a highly incremental way and designed for RDB2RDF data integration. At the same time it is far less automatic than our own approach. In particular, Karma makes suggestions for mapping *semantic types* inside their editor, which the system learns from previously integrated types. That is, their incremental steps occur between repeated mappings of source types to the same semantic concepts. Karma will make suggestions mostly in multi-source integration but will usually not start to provide any suggestions before at least one source has been manually integrated. Also, certain mapping steps in Karma provide no suggestion support at all and will always require elaborated manual input in the form of transformations scripted in Python.

Bernstein et al. describe a demonstration of a pay-as-you-go schema matching approach that they also call an incremental approach [15]. Basically, the authors argue from a HCI point of view that users are overwhelmed when confronted with automatically calculated matches for the whole schema at once and thus need to proceed step by step. Also, they include an “implicit scope” in their match generator, which favors matches in areas of other recent matches. On the side of match generation and mapping composition no aspects about the incremental state of mappings are leveraged or even discussed. In terms of matching and mapping the approach is therefore rather just interactive than incremental.

Wagner et al. have proposed a thoroughly incremental ontology aligner, the I3M aligner [74]. The tool works both interactively and incrementally over several iterations (hence the third 'I' in the name of their aligner). The key idea is to split ontologies into *partitions*, and match each partition in several iterations, using different matchers and collecting user feedback in-between iterations.

In a way very similar to incremental mapping, Lambrix and Liu analyze the benefits of using what they call “partial reference alignments” for ontologies [75]. Partial reference alignments refer to confirmed partial mappings between two ontologies. In contrast to a truly incremental approach, the authors consider partial mappings only as initial input in one single processing step. They make use of such input in three different ways. First, they override any other suggestion with confirmed partial mappings, where available. Second, they consider using partial mappings to partition the mapping problem into smaller bits to improve scalability. Finally, they use given references in the matching step, following the observation, that often “common patterns can be found between two correct mappings”. Their observed patterns are linguistic patterns on a string level, i.e., they describe similarities between the linguistic structure of term strings. The authors also experiment with all of the above mentioned techniques in filter step during mapping selection. Overall, they discovered only minor advantages of using partial mappings in their series of experiments.

QODI [20] presents an approach to use queries over an ontology target schema to improve the choice between ambiguous mappings. The approach could in principle fit into an incremental mapping generation process driven by queries. However, the authors' main concern in the paper is to disambiguate alternative mappings at runtime in the context of queries. QODI and its semi-automatic commercial sister system Ultrawrap Mapper [22] have been built for use with OBDA system Ultrawrap [76].

The COMA system [16, 24] is designed to be interactive and iterative as a result of the observation that fully automated solutions are not usually feasible. A fully automated, one-iteration mode exists as well. Through re-ranking and reuse of mappings

between iterations it is also partially incremental. However, it does reuse mappings only in wider settings where a larger number of schemata comes into play (or a number of similar chunks of a schema, called *schema fragments*). Reusability of individual previous matches is based on the similarity of the schemata or schema fragments with matched schemata or schema fragments, respectively. It is then assumed, that matches can be interpreted transitively between the different schemata. Essentially, the matcher performs a join on two transitively connected matches, thus deriving a third one, which directly connects the first schema element to the last. This presumably yields the best results when several schemata are to be matched pairwise in all directions. It has arguable no or little effect on basic schema translations from one source to one target. At the same time, user feedback can be employed to accept and reject match candidates between iterations. Similarity scores for accepted or rejected nodes are adjusted and remain invariant during follow-up iterations. This corresponds to one of the three strategies in IncMap for leveraging user feedback. For the interactive and iterative mode of COMA, no evaluations have been performed in [16].

In terms of interaction paradigms, semi-automatic approaches can either follow the traditional semi-automatic workflow (i.e., present many correspondences and allow to remove, add, and edit them), or they could assist users in with suggestions in a context where they are potentially useful. Such a context could be, most obviously, a mapping editor, where an expert user has already started to create some mapping. It could also be some more sophisticated and indirect context, e.g., a view that requires additional mappings in virtual integration, or some sort of a hybrid approach.

At the heart of understanding opportunities for mapping suggestion support beyond traditional semi-automatic matching are mapping editors. For RDB2RDF, the standard mapping language is R2RML [77], which has recently received the status of a W3C recommendation. Some existing RDB2RDF editors offer advanced and more or less visual user interfaces, e.g., [78, 79]. However, these are based on domain specific languages predating R2RML. [80] describes an Eclipse plugin that supports R2RML execution as well as mapping generation with custom algorithms. Neto et al. have demonstrated a mapping editor with a highly visual interface that eventually generates R2RML [81]. However, they do not expose R2RML semantics but only simple correspondences used as assertions. Arguably, the expressiveness of these assertions is only a subset of R2RML. Our basic R2RML editor [82] originally demonstrated in a much earlier version as [83] closely follows the syntax and structure of the R2RML language. Users can visually edit complex mapping rules in structured views, where each view corresponds to one top-level R2RML rule (called a *TriplesMap*). This leaves room for mapping suggestion in a per-rule context and at various degrees of rule completion.

A few recent papers include user studies that analyze approaches for user-centric data integration and thus take a broader look at interface support options, also for built-in mapping suggestions.

Falconer and Noy summarize and discuss the state of the art in semi-automatic ontology matching and visualization [84]. To the authors, this largely means any kind of user involvement on the one hand and explanation of results on the other. Besides that, they consider crowd sourcing and Web 2.0-style collaborative user involvement as a related challenge. While generally assuming the traditional semi-automatic approach as a baseline, where matches are first produced automatically and then adjusted and complemented manually, the authors note an increasing “trend towards a more human-centered approach to ontology matching” [84, p. 30]. In this context, they talk about an upcoming “symbiosis between tool and user” that gives rise to a growing number of new tools, following this novel trend. In their view, a key reason for this trend are problems that arise from large ontologies and large sets and the large number of resulting correspondences produced by tools in the traditional approach. The paper surveys interactive ontology alignment tools (e.g., [24, 57]) in some detail and also briefly discusses traditional schema matching tools. In their discussion, the authors note that there are important principled differences between schema matching and ontology alignment, which mostly stem from the different rationales when designing ontologies vs. relational schemata. This supports our argument that the specific requirements of RDB2RDF mappings call for dedicated, specialized tools. Evaluations of interactive approaches are briefly discussed in the paper and the authors summarize five (mostly small-scale) user studies.

Stuckenschmidt et al. have fielded a user study [85] that analyzes the effectiveness and efficiency of their own interactive approach and follows a cognitive support model [86]. This cognitive support model defines a fairly general and principled approach for creating mapping rules. Here, a set of mapping rules is first created automatically, then these rules are applied to some data, and finally users verify the individual rules by marking the results as correct or incorrect.

In contrast to this strict procedural approach, Wrangler [87] introduces a new visual and interactive data transformation language that leaves much freedom to the user as to which approach they will apply. Basically, the user can apply a set of data transformation primitives in any order and is supported by interactive data visualization tools to preview results, histories to undo changes, etc. Rather than schema mappings the tool is built with data matching and transformation in mind. Wrangler focuses on ease of use and clarity of transformations, discussing advanced HCI aspects, transformation documentation and provenance. Natural language descriptions of transformations and

visualized previews also play an important role. Implementation-wise, an important feature of the system is a dual execution strategy, allowing both online previews and in-browser transformations as well as compilations of the same rules into Python scripts or even MapReduce [88]. The authors also present a user study comparing Wrangler with manual Excel transformations, where human effort to a perceived final solution of each task is the measure of success.

### 2.3.3 Inter-Model Mapping

Inter-model mapping, i.e., mapping from a schema in one data model to another schema in an entirely different data model, introduces additional challenges. All popular data models are based in logic. At the first glance, those additional challenges could therefore appear to consist simply of the transformation of modeling primitives and modeling structure. At second glance, however, this is not the case. Besides the structural transformation, inter-model mappings lead up to at least three additional road blocks that are not present in intra-model mappings:

1. Impedance mismatch, i.e., differences in how the model relates to its data.
2. Different expressiveness, e.g., the logically formulated semantics in OWL ontologies are much stronger and expressive than semantics of relational algebra.
3. Different purpose and usage, i.e., different goals w.r.t. the kind of data that should be expressible in the model and about how that data might typically be used.

Impedance mismatch, a term borrowed from electrical engineering, has initially been used in data management with object oriented databases as the object-relational mismatch. The object-relational mismatch states that problems arise because the relational model structures data as related data *values*, while other models structure data as *objects*. For the wider object-relational mismatch see [89], for a discussion in the context of RDB2RDF c.f. [90]. Also, another side of the impedance mismatch for RDB2RDF mappings is the gap between the closed world assumption and the open world assumption. The modeling of some piece of information might be perfectly clear in closed-world semantics, but be less clear (or at least incomplete) in open world semantics. For instance, this has lead for some systems to require mappings to be marked as *exact*, when it is known that all relevant information is covered by the mapping in accordance with closed world semantics (e.g., [79]).

In addition, different levels of expressiveness are supported in different data models. Although the issue is a general problem in finding semantic mappings [1, Chap. 3]

whenever application semantics play a role, it becomes more pressing when models involved already enforce a gap in semantics.

Finally, a more practical problem originates from how experts are *used to* design data in each model. This has to do with broadly accepted design practices but also with the typical purpose of modeled data. For instance, relational data should likely be queried in many ways and queries should perform efficiently. Ontological data might be used for reasoning and interlinking of information.

Inter-model mappings have first been discussed between relational databases and object oriented databases [91], later for XML. Systems built for both relational databases and XML include many of the later tools (e.g., Clio [55] and, of course, more general-purpose matchers such as (e.g., COMA [16]). They usually feature inter-model mappings, but offer no or little specific optimizations for the case w.r.t. the above mentioned challenges. More recently, RDB2RDF mappings have become a topic of increased interest.

In either case, relational databases are often on the source side of the mapping. One technique that is frequently employed with relational databases to close in on the expressiveness gap is database reverse engineering. Database reverse engineering is a field of primarily earlier database research that attempts to reconstruct conceptual models from logical or even physical relational database models. Some of the more prominent early papers include [92, 93]. Traditional reverse database engineering is usually aiming to construct entity-relationship models (ER) or extended entity relationship models (EER) from a logical relational schema.

Müller et al. summarize database reverse engineering tools and techniques from the perspective of the year 2000 as well as some of the key challenges of the time [94, Sec. 3]. In their paper, they also put database reverse engineering in the context of wider reverse engineering motivations, techniques and challenges. Malpani et al. propose a modern approach to database reverse engineering [95]. They follow a more recent motivation for the task, by connecting the idea of understanding database content with the task of building applications for the data. This is not unlike the motivation for OBDA, which often originates from the desire to use domain ontologies as a semantic, conceptual basis to construct complex queries in the domain.

Reverse engineering techniques have also been occasionally discussed in the context of RDB2RDF (e.g., [96, 97]). Many more papers effectively consider the same class of techniques, but usually without alluding to database reverse engineering explicitly (e.g., [19, 21, 23, 98]). In IncMap, we also effectively rely on database reverse engineering techniques by recognizing modeling patterns. However, we do not use these patterns to *transform* the database schema, but to *annotate* its matching graph, s.t., both the

original structure and recognized patterns could be exploited for mapping generation. Also, our patterns are based on previous enumerations of RDB $\rightarrow$ RDF mapping patterns ([98]) together with relational design patterns as explained in database textbooks (e.g., [28, Chap. 4.5f]) and typically taught in undergraduate database courses.

For XML, semi-structured data and unstructured data, the process corresponding to database reverse engineering is often referred to as *schema extraction* [99]. This term reflects the fact that data with less rigid structure does not necessarily originate from an engineered model translation process that could be reversed.

RDB $\rightarrow$ RDF mappings form a case where the inter-model gap is particularly wide. The general groundwork has been laid out by RDB $\rightarrow$ RDF mapping languages and transformation systems. Popular established systems include D2R [100], or R<sub>2</sub>O [101]. As mapping language, R2RML [77] has recently become a standard. For a survey on mapping languages that predate R2RML, c.f. [102]. Some recent research has also suggested alternatives to R2RML, e.g., [103], or [104], which extends R2RML to map also map from non-relational sources to RDF.

Most closely related to i<sup>3</sup>MAGE, some previous work has described efforts to automatically construct RDB $\rightarrow$ RDF mappings. In practice, however, most systems do not approach the task of true inter-model mappings. Instead, they attempt to transform the problem into a better understood, yet not equivalent problem, e.g., ontology alignment [65]. For example, [105] transforms relational schemata and ontologies into directed labeled graphs and reuses COMA [24] for what essentially amounts to syntactic graph matching. The few approaches for directly matching aspects from relational schemata to corresponding aspects in ontologies date back several years and have been written with a different motivation and under vastly different preliminaries. Overall, the driving motivation to develop automatic and semi-automatic RDB $\rightarrow$ RDF techniques at the time can be summarized as the desire to get more data into the Semantic Web (in a most literal sense, i.e., to complement actual web pages).

KAON-REVERSE [106] represents one of the earliest exceptions. The system is designed to directly map relational data to a given ontology and explicitly uses database reverse engineering [92, 107]. As a target language, F-Logic [89] is being used, which was initially developed as a formal logical language for object oriented databases [91]. The authors provide a tool for semi-automatic usage, i.e., with mapping suggestions. To produce suggestion it uses a number of fixed mapping rules defined in the paper, which they derive from reverse engineering of the logical relational schema. Those, in turn, are based on a reverse-engineered semantic representation based on the database logical schema. The reverse engineering process borrows from earlier approaches of mapping relational databases to object oriented databases. In an unusual trait, user interaction

for semi-automatic mapping happens on two separate stages, reverse engineering and concept alignment. Without human interaction, both reverse engineering and automatic mapping act deterministically and in ignorance of each other on ambiguous patterns. In this case, although formally following a direct mapping approach, KAON-REVERSE behaves very similar to an indirect mapping approach: the reverse engineering phase could be considered an initial mapping, while automatic mapping corresponds to plain ontology alignment.

Another early system, RONTO [23], uses a combination of syntactic strategies to discover mappings by distinguishing the types of entities in relational schemata. The tool is aimed explicitly towards semi-automatic mapping generation of relational schemata to ontologies. In their paper, the authors also discuss the issue of bootstrapping ontologies for the purpose. RONTO uses basic rules to match concepts and properties to relations (including views) and attributes. The system follows a direct inter-model mapping approach, which enables the use of inter-model mapping patterns. More specifically, RONTO considers  $n : m$  relationship relations and joined tables.

Hu and Qu exploit structure of ontologies and relational schemata by calculating the confidence measures between virtual documents corresponding to them via the TF/IDF model [66]. The authors maintain that any purely manual approach to constructing mappings would be tedious and therefore improbable, thus also assume a semi-automatic approach.

Finally, An et al. describe an approach to derive complex correspondences for a relational schema to ontology mapping using simple correspondences as input [108]. The paper mentions the problem of different design patterns used in ontologies and relational databases, but stops short of addressing the issue of patterns specifically. Instead, the authors focus on a special case to follow their primary aim of deriving complex mappings.

Another avenue of work considers mapping generation, but primarily in the sense of mapping and ontology bootstrapping, i.e., they produce mappings and a fitting ontology based only on the input schema and without considering a target schema.

For instance, Sequeda et al. [31] have formally defined a direct RDB $\rightarrow$ RDF mapping, based on the general ideas behind W3C's definition of a direct mapping [109]. The basic motivation is to automatically generate a mapping together with a target ontology. In their paper, the authors base their direct mapping ontology on RDF and OWL vocabulary. As specific contributions, they prove that selected basic and desirable properties hold for direct mappings following their approach. In particular, they can guarantee information preservation (i.e., any database could be losslessly restored from the generated graph by an inverse mapping) and query preservation (i.e., all queries could be

equivalently translated) as well as either one of monotonicity (i.e., adding data to the source database has no effect on triples generated for any previous data) or semantics preservation (i.e., referential integrity should be truthfully reflected in the resulting RDF graph's consistency). The authors prove that monotonicity and preservation of semantics are mutually exclusive, and no such direct mapping could guarantee both.

Other examples for bootstrapping mapping generators include [21, 76, 79, 110]. In principle, BootOX [19] also fits into that category, but in contrast to the others it is built to cooperate with an ontology aligner to eventually still map to a target ontology.

A comprehensive overview of RDB2RDF efforts, including (but not limited to) related approaches of automatic mapping generation, can be found in the survey of Spanos et al. [90].

### 2.3.4 Mapping Quality and Benchmarking

Mappings between ontologies are usually evaluated only on the basis of their underlying correspondences (i.e., *alignments*). The OAEI [25, 111] provides tests and benchmarks for those alignments that can be considered as a de-facto standard, although alternative benchmarks have also been proposed [112]. OAEI results are being published regularly (e.g., [26]).

However, other related benchmarks have also been proposed and approach the topic from a different angle. For instance, Rivero et al. [113] have devised a benchmark for linked data translation (data exchange), i.e., for mapping RDF data on the web between different vocabularies. There, the authors assume a particular mapping model in which all mappings are based on correspondences in such a way that *executable mappings* are queries resulting directly from a combination of correspondences. Measured aspects include expressivity (i.e., the general ability of systems to somehow express either of the mapping patterns) and also performance. The benchmark does not, however, consider any kind of precision or recall of produced mappings besides the upper limit given by expressivity.

For XML mappings, at least one somewhat influential benchmark, STBenchmark [114], has been proposed. STBenchmark has been designed for XML but has a potential reach beyond the XML model, as it supports a number of fairly generic mapping challenges. However, its generated data do not support any of the specific properties of models other than XML, e.g., no relational constraints or ontology axioms adding advanced expressivity. STBenchmark offers capabilities for generating pairs of source and target XML schemata in a controlled fashion as well as instance data and XQuery reference

mappings. It uses fairly generic terms for mapping challenges (or *basic mapping scenarios* in STBenchmark terminology). For instance, normalization artifacts are called vertical partitioning, while the hierarchy pattern that requires us to extract several entity types from a single table is called horizontal partitioning. STBenchmark does not outline any single one benchmark but is rather considered a mapping benchmark scenario generator.

Mappings between relational databases are typically not evaluated in a common benchmark. Instead, authors compare their tools to an industry standard system (e.g., [16, 17, 24, 55]) in a scenario of their choice. However, some benchmarks have also been proposed (e.g., [115, 116]).

In addition, certain non-comprehensive evaluation criteria have been presented as side contributions in other work (e.g., [117, 118]).

A novel TPC benchmark, TPC-DI [119] was created only recently. It is based on a fixed set of source and target schemata. TPC-DI has a clear focus on the wider field of data integration as opposed to schema mappings, only. Notably, however, this includes the integration of heterogeneous sources comprising relational and XML data sources and thus also touching aspects of inter-model data integration. Aiming to be an industry class benchmark, TPC-DI is designed to assume a typical industry setup. For instance, the target schema is a classical relational OLAP data warehousing schema structured around dimensions and fact tables. As of March 2016 no results have been published on their web site,<sup>6</sup> though. Also, TPC-DI is designed mostly to benchmark ETL mapping execution efficiency, as opposed to mapping quality.

Similarly, evaluations of RDB2RDF mapping generating systems were earlier based on one or several data sets deemed appropriate by the authors and are therefore not comparable. In one of the most comprehensive previous evaluations, QODI [20] was evaluated on several real-world data sets, though some of the reference mappings were rather simple. Also, IncMap [52], a component of i<sup>3</sup>MAGE, was initially evaluated on real-world mapping problems based on data from two different domains.

A number of papers discuss various quality aspects of generated mappings in a more general manner.

Console and Lenzerini have devised a series of theoretical quality checks w.r.t. consistency [120]. Bienvenu and Rosati discuss query-based notions of mapping equivalence in OBDA from a formal point of view [121]. Westphal et al. have discussed a wide range of possible quality measures that may play a role in for RDB2RDF [122]. In a similarly broad approach, Wang has proposed a framework for quality assessment in data integration [123].

---

<sup>6</sup><http://www.tpc.org/tpcdi>

In a related benchmark, Impraliou et al. generate synthetic queries to measure the correctness and completeness of OBDA query rewriting [124]. The presence of complete and correct mappings is a prerequisite to their approach. Mora and Corcho discuss issues and possible solutions to benchmark the query rewriting step in OBDA systems [125]. Mappings are supposed to be given as immutable input. The NPD benchmark [126] measures performance of OBDA query evaluation. Neither of these papers, however, addresses the issue of systematically measuring mapping quality w.r.t. utility, i.e., expected results. In an even more specific variation, Schoenfish and Stuckenschmidt propose a benchmark to measure the performance of OBDA systems at scale using probabilistic databases and ontologies [127].

Our RODI benchmark suite [128] has been developed specifically to fill this gap and systematically evaluate the utility of i<sup>3</sup>MAGE, also comparing it to competing approaches.

An alternative benchmark that could be used in a way similar to RODI is iBench [116], which has most recently been presented in the form of a technical report and follows the idea sketched out in another, earlier technical report [129]. While iBench provides a significant degree of control over synthetic data generation, it does not offer a complete benchmark framework for ready evaluation of RDB $\rightarrow$ RDF tools.

In iBench, the authors present a meta-data generator for benchmarking schema mappings. Essentially, the idea is to automatically generate rich schemata and also logical GLAV mappings between them in a scalable fashion. iBench is designed as an extension of an earlier benchmark, STBenchmark [114]. Just as in STBenchmark, the authors consider their work as a basis to create different benchmarks in data integration and related fields rather than one fixed benchmark. The generator is constructed to produce scenarios in two steps: first, *simple* scenarios are generated according to specifications. Simple scenarios consist of one source and target schema each. An *orchestration engine* could then chain and combine several such simple scenarios into complex ones. And second, the orchestration engine can trigger a data generator to fill schemata or produce a sample query workload. As primitives, iBench aims to support both handcrafted specific challenges as well as wrapped-up real-world examples. Notably, iBench extends STBenchmark to fully support relational models, e.g., by supporting the generation of database constraints but also by supporting additional mapping patterns, called *mapping primitives* in the paper.

In their papers, the authors discuss certain details of fundamentally different use cases and setups that may call for the generation of different integration benchmarks, e.g., schema evolution mapping, mapping composition, initial mapping creation, etc. The authors also give a formal definition for the requirements of generating such meta-data

(i.e., to generate schemata) and for the relations of meta-data elements in different schemata.

The later report [116] also provides an initial evaluation with three mapping generating systems, including Clio [17]. Notably, in this evaluation, the authors include an RDB2RDF mapping scenario. However, they build this scenario in a mostly naive fashion. In particular, they consider only four specific RDB2RDF mapping patterns and give little consideration to discussing their choice of patterns. It seems that their aim is to use a basic set of patterns that may *possibly* be used to map *some* relational databases to an ontology. For instance, they employ exactly one hierarchy pattern, called *IS-A*, which assumes one single modeling pattern of hierarchies in all relational databases. Intuitively, this mindset for bridging the inter-model gap is similar to the one assumed by the basic version of IncMap without patterns and reasoning. However, the scenario demonstrates how iBench might in the future be used to generate benchmarks similar to RODI, assuming a thoughtful selection of additional patterns and a fitting configuration. In any case, iBench does not produce actual ontologies in any standard language (such as OWL ontologies) or reference mapping (like R2RML mappings) but only an XML description, which indirectly mimics such artifacts using a relational metaphor and terminology.

A survey by Bellahsene et al. gives an overview of different approaches for evaluating schema matching and mapping [130]. The paper also includes an overview of system characteristics that enable or even call for those different types of evaluation. In their notion all data models, including ontologies, are sufficiently described by structural schemata, and thus no distinction between different models needs to be made w.r.t. schema matching and mapping. Consequently, the survey does not specifically cover RDB2RDF or any other inter-model mappings. The authors start out by emphasizing the importance of evaluation standards for the increasing number of mapping tools. While observing that no generally accepted benchmark for schema matching and mapping did exist at the point, they identified different metrics that are recurrently being used for evaluation. Among those, they singled out quality metrics based on precision and recall as the key measures for efficacy, including the kind of end-to-end instance comparisons now used in RODI. More theoretical quality measures as well as efficiency measures that cover both computational efficiency and human effort are also discussed in the survey.

## Chapter 3

# i<sup>3</sup>MAGE Approach

In this chapter, we give a systematic overview of our approach with all relevant main aspects. We describe in Section 3.1 the key problem addressed and list associated challenges. Then, we introduce the overall idea of our proposed approach, and present the rationale behind choosing this particular approach in Section 3.2.

### 3.1 Problem Statement and Challenges

#### 3.1.1 Problem Statement

i<sup>3</sup>MAGE addresses inter-model mapping tasks for RDB $\mathcal{2}$ RDF in an incremental and interactive fashion. That is, it attempts to generate and improve mappings between a given relational source schema and a given target ontology with support of user feedback over several semi-automatic iterations. Formally, i<sup>3</sup>MAGE rewrites input mapping instances with initial (and possibly empty) sets of mapping rules in the shape of  $I_{in} = (R, O, M_{in})$  into output mapping instances  $I_{out} = (R, O, M_{out})$ , with  $R$  a relational source schema,  $O$  a target ontology and  $M_{in}, M_{out}$  sets of mapping rules. The aim of rewriting is to increase utility of the mapping instance (c.f. Definition 2.8). i<sup>3</sup>MAGE needs to implement an incremental approach by running any number of iterations rewriting the output of the previous iterations  $I_{in}$ . It also supports interactive feedback by accepting user feedback as additional input in-between iterations.

The primary research problem is therefore to design a mapping generator (c.f. Definition 2.7), which is suitably adapted to the specific challenges of RDB $\mathcal{2}$ RDF inter-model mappings, as opposed to regular intra-model mappings.

With what amounts to a second problem to solve, the system is required to make use of semi-automatic feedback in this specific context as effectively as possible.

In particular, this means that, unlike most other semi-automatic mapping generation approaches, *i<sup>3</sup>MAGE* needs to work fully *incrementally*. That is, *i<sup>3</sup>MAGE* should map only parts of the schemata at a time and needs to allow manual modifications between any two incremental steps. It should also be capable of leveraging previous partial mappings resulting from such modifications to improve the quality of subsequently generated mapping parts. User feedback should also be employed where available to re-rank and refine results, making the approach semi-automatic. In addition, information needs that stir the incremental process should also be leveraged as context.

### 3.1.2 Challenges

As *i<sup>3</sup>MAGE* is targeted to RDB2RDF inter-model mappings, specific challenges result from the inter-model gap. While all kinds of mappings are challenged by the modeling differences between schemata, this is particularly true for inter-model mappings. In addition to common issues such as slightly different semantic perspectives (and thus modeling) of the same aspects, inter-model mappings have to deal with different modeling primitives on both ends.

For RDB2RDF, the relational source models consist of constrained relations between values, while the target ontology models contain a semantically rich description of connections between the entities of semantic concepts, their meaning and possible implications. Largely different modeling primitives and different degrees of expressiveness on either side, however, lead to the use of thoroughly different modeling patterns.

This situation is aggravated by the different perspectives that database architects and ontologists take on their respective data models: relational database schemata are seen as the technical, low-level models that they are. Consequently, they should host data in a clean but most of all in a high-performing and usable way. Ontologies, on the other side, are mostly considered from by a high-level, meaning-driven point of view. In other words, while a database architect aims to fit existing or required data somehow into a model and may restructure them following technical rules with little consideration of data semantics, an ontology designer attempts to describe the meaning of both present and potential future data as accurately as possible. For example, a database architect might put two semantically distinct concepts together in one table if they are in a strict 1 : 1 relation with one another. They would also split up information about one single concept into several tables if some of its properties were in a  $n : 1$  relationship with the concept.

The main challenge in RDB2RDF mapping generation vis-a-vis regular, intra-model mapping approaches is thus to understand and handle the different modeling patterns appropriately.

In addition, *i<sup>3</sup>MAGE* stands apart from many semi-automatic mapping approaches in being fully incremental, i.e., the system does not only accept manual input at the end of the process but effectively and continuously reacts and adjusts to user feedback and input while producing the mapping piece by piece. This process can also be driven by immediate information needs (i.e., queries), which could serve as additional input.

However, compared to different modeling primitives and patterns, these aspects play a lesser role when it comes to specific challenges. Mostly, user feedback is solicited in a form similar to other, well-studied mapping generation approaches. Thus, the use of existing partial mappings and pay-as-you-go query workload as input are left as the only challenges to the system that significantly exceed mere engineering problems on the semi-automatic side.

### 3.1.2.1 RDB2RDF Challenges Regarding Modeling Primitives

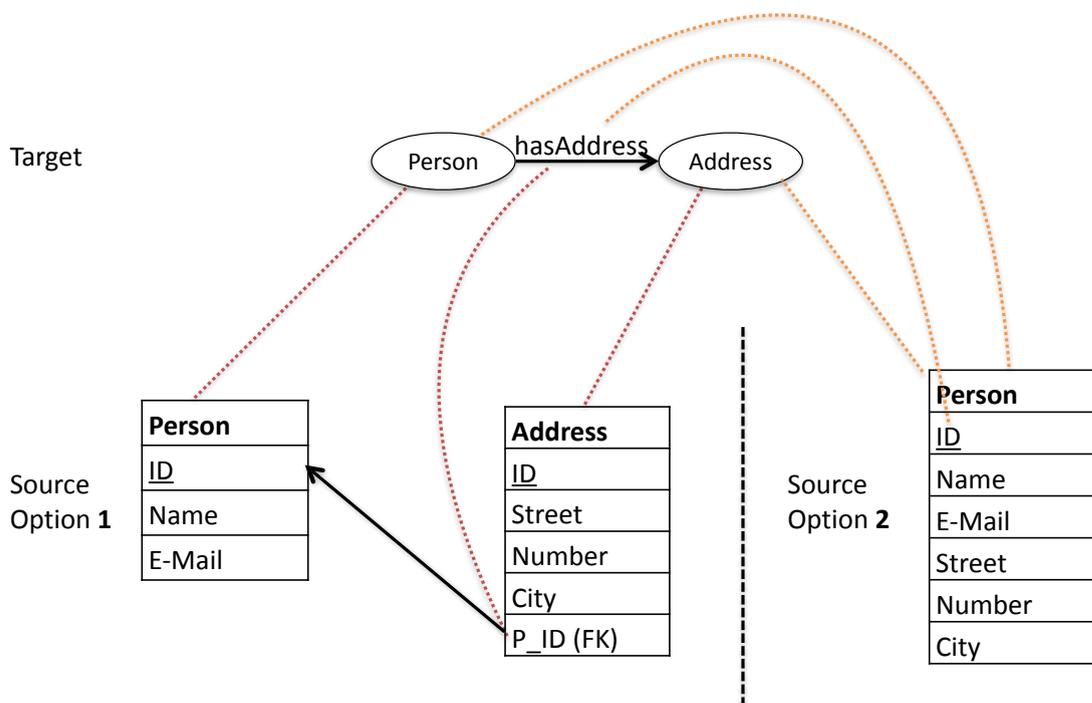


FIGURE 3.1: Simple inter-model matching scenario with persons and their addresses (target ontology, and two alternative relational source schemata, including matches)

In inter-model mapping, even basic modeling primitives display a number of differences that are no concern for intra-model mappings.

**Example 3.1** (Persons, Addresses). *Consider a simple ontology with persons and their addresses, together with different fitting database schemata as shown in Figure 3.1. The ontology consists of classes Person and Address. In addition, there is an object property, hasAddress, that connects those concepts. Datatype properties are not depicted for brevity.*

Figure 3.1 illustrates a mapping scenario for this example. Matches in this scenario are considered for classes and object properties with tables and foreign key constraints. Although still rather trivial, even this simple example illustrates some of the inter-model gap encountered in inter-model matching.

In case of the first (left hand) source schema, the two classes match with tables while the object property matches with a foreign key constraint. While most clear and straight-forward, it hides a first aspects of the inter-model gap: in fact, the relation described by object property *hasAddress* from the ontology is not represented by a foreign key in the relational database but by a join path established through a join condition (or join predicate) between the two tables. What the foreign key gives us is simply a hint on how to phrase such a reasonable join condition. In general, however, a join condition is something, which can be formulated freely *on top of* a schema. It can not be found anywhere *inside* the schema explicitly and thus cannot easily be used as an element in schema matching. Even in this straight-forward case, slight changes in modeling could make this issue visible. For instance, if the foreign key was not explicitly modeled in the schema, a straight-forward match between schema elements (i.e., between a property and a constraint) would no longer be possible.

For the case of the second (right hand) source schema in Figure 3.1, we assume that persons and their addresses are related 1 : 1. This can be a reasonable assumption for a relational database if only one (i.e., primary) address will be kept per person anyways. In this case, relational design theory recommends to join all information about both entities into a single table. Both classes from the ontology can still match with a table (the only table in the schema) in a  $n : 1$  matching. There is no clear match for object property *hasAddress*, though. Neither of this would be a problem in intra-model mappings, e.g., ontology alignment or relational schema matching. For ontologies, relations between different entities will always be modeled as object properties on either side. In relational schemata, on the other side, relations and join conditions that express them exist independently from the question of how the data is managed in different tables, as long as functional dependencies are preserved. Therefore, establishing those connections in a matching is less critical.

Although these differences between modeling primitives require some consideration when matching, they do not pose a novel research challenge to this point. At the end of the day, relevant differences are relatively clear and non-ambiguous. Therefore, they can be easily handled individually by enumerating reasonable match types of different modeling primitives (e.g., RONT0 [23]). Also, differences could be neutralized in a unified internal representation (e.g., COMA [16]).

### 3.1.2.2 Basic Mapping Pattern Challenges

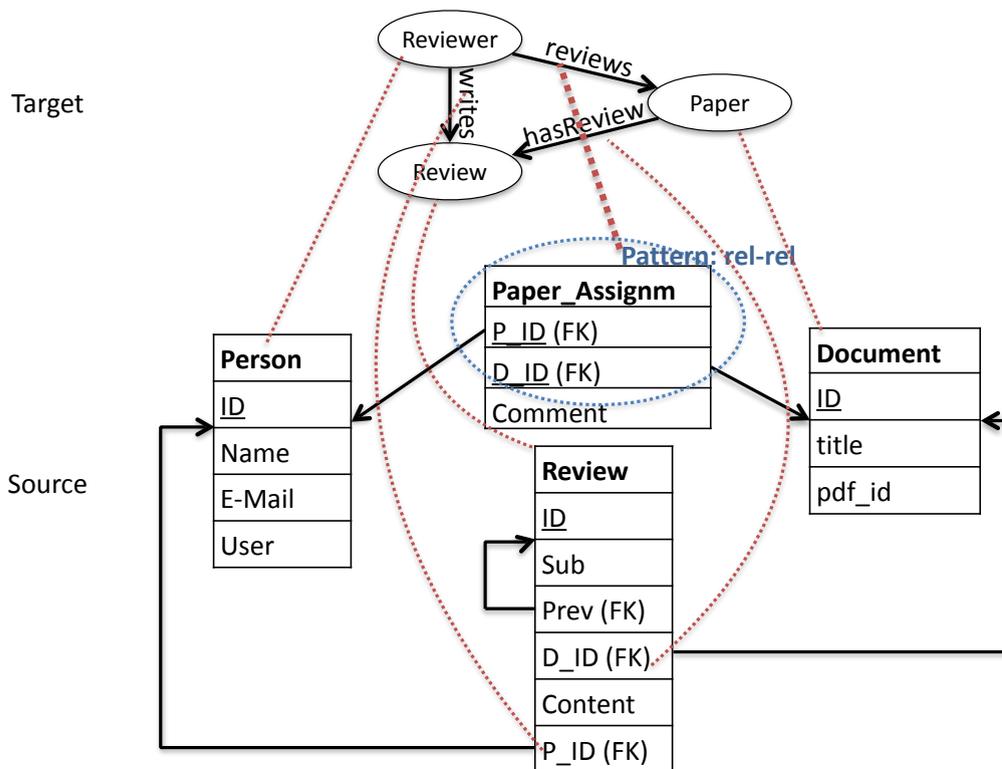


FIGURE 3.2: Inter-model matching scenario with papers, reviews and reviewers

Modeling patterns in inter-model mapping complicate the situation beyond mere modeling primitives. This is the case even for relatively simple and highly frequent patterns, such as the ones used in relational databases for modeling  $n : m$  relationships or  $1 : n$  datatype properties.

**Example 3.2** (Papers, Reviewers, Reviews). *Consider a small ontology in the conference domain with papers and associated reviews. The ontology consists of classes Paper, Reviewer and Review. In addition, there are object properties that connect those concepts: a reviewer reviews a paper and thereby writes a review. Also, papers may have reviews (hasReview). Datatype properties are left out for brevity.*

Figure 3.2 shows both this ontology and a matching relational database schema.

The example includes one highly common modeling pattern from relational databases, a *relationship relation* (*rel-rel*; c.f. [27]). Relationship relations are the default method to model  $n : m$  relationships between entity types in the relational world. The correct correspondence for this pattern in the ontology would be the property *reviews*. However, no single modeling primitive in the database schema is a suitable match for this property. It is neither table *Paper\_Assignm*, nor any single one of its attributes that describe the connection. Instead, what the corresponding object property does express is a join path between document and person that involves two separate attributes in *Paper\_Assignm*, in addition to key attributes from both tables that it connects.

Generally, the database schema in this example is more realistic and a bit more complex than the previous example in terms of relational modeling patterns. Still, most systems get this right in some way, as this particular pattern of relationship relations is so obviously important that no RDB2RDF mapping generation system can expect to produce reasonable output without it. For instance, COMA [16] considers this one specific pattern by supporting (and identifying) matches between object properties and tables that in fact implement the pattern of relationship relations. A subsequent mapping generator then needs to take the hint and construct the join predicate accordingly.

Numerous such patterns exist, especially on properties and type hierarchies. For instance, besides the above example there are different patterns for  $1 : n$  datatype properties. Also, subclass relationships can be commonly modeled in at least three different ways in relational schemata.

The general importance of considering mapping patterns for realistic mapping generation has long been known and accepted in the wider field of schema mapping. For instance, STBenchmark [114] is built around a series of complex mapping patterns within XML, named *basic mapping scenarios*.<sup>1</sup> More recently, iBench [116] even considers some primitive inter-model patterns as an unavoidable means to test the quality of mapping generators (although mapping patterns are dubbed *mapping primitives* in the report). For RDB2RDF, Sequeda et al. enumerate a significant subset of fundamental mapping patterns [98].

### 3.1.2.3 Advanced Inter-model Mapping Patterns

More advanced patterns exist as well. And while most RDB2RDF systems can still handle some of the most basic patterns, this is not necessarily the case for any of the more advanced ones. This is because most traditional systems do not consider patterns

<sup>1</sup><http://db.disi.unitn.eu/pages/stbenchmark/basic.htm>

systematically, and instead implement a special-case treatment for some of the most obvious cases, such as  $n : m$  relationship relations.

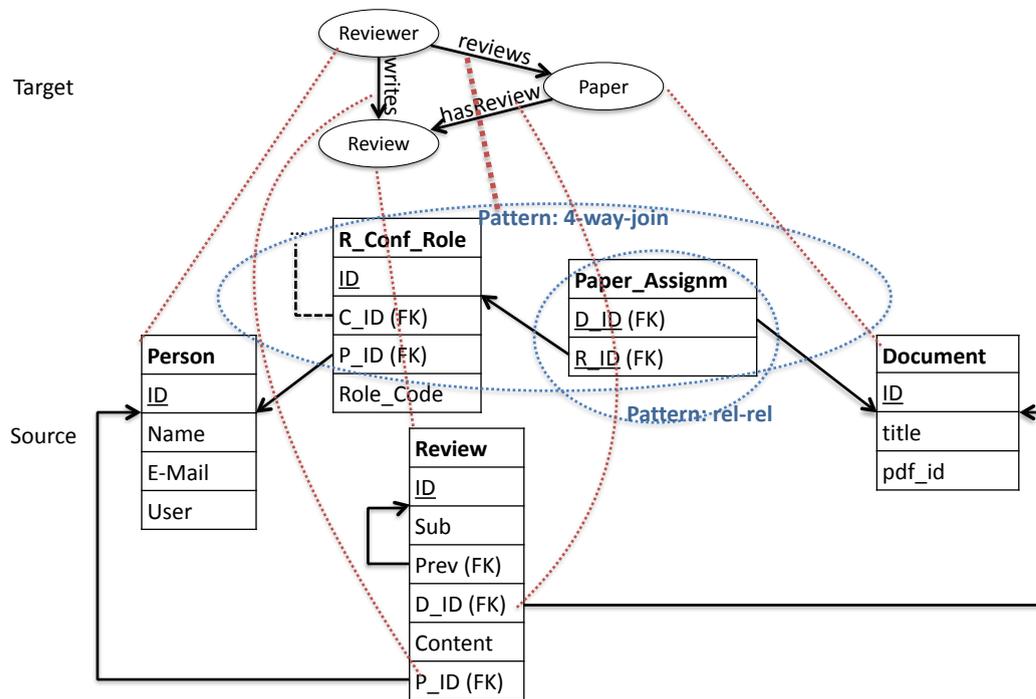


FIGURE 3.3: Extended inter-model matching scenario using advanced patterns

Still following Example 3.2 with papers, reviewers and reviews, Figure 3.3 shows a different database schema with the same ontology as before.

The schema differs from the previous one by having an even more indirect relationship between tables *Person* and *Document*. An intermediate relation (*R\_Conf\_Role*) has been added, which introduces a role of applicable persons. For instance, a paper could be assigned to John in his role as a reviewer of some conference, but also to Jane in her role as track chair at the same conference. This pattern essentially represents lazy modeling of an  $n$ -ary relationship. Connecting *Person* and *Document* correctly now requires a four-way join. Although both less likely and more ambiguous than the plain  $n : m$  relationship depicted previously (Figure 3.2), this is a pattern that occurs in practice. And in case of the current example it is required for the only semantically correct interpretation for a mapping to the target ontology.

One critical aspect in this example is the ambiguous nature of the pattern: while semantically intuitive, there is no strong technical evidence that would turn it into an obvious intermediate link between persons and documents. Also, following the common assumption of Steiner trees, i.e., that the shortest path between any two tables is also most likely the correct semantic connection between them, would lead to an incorrect interpretation in this case (c.f. [108]).

It is thus important to understand and consider this pattern in a the way that it *could* point to a reasonable match in this case, but with a good degree of uncertainty. It is similarly important to understand the pattern to be what modeling patterns are by nature in the general case: a design decision that follows certain principles, but which is also influenced by the preferences, experience and convenience of whoever makes the design choices. Therefore, it is a reasonable assumption that non-trivial patterns have a differently likelihood in different schemata, depending on who did design the schema and to which purpose. Consequently, it appears insufficient to use only fixed if-then rules to produce matches from such patterns.

More, and more complex patterns of that sort exist in everyday database design. Examples include modeling of complex or indirect relationships, various modeling patterns used for type hierarchies, recursion, or even symbolic encoding of some significant individuals for partitioning.

#### 3.1.2.4 Approaches to Pattern Recognition Challenge

Quite generally, mapping generation between schemata of different models can be approached in two different ways: (1) directly, by relating the schemata of different models with one another using some internal representation that allows to identify such direct matches, or (2) indirectly, by first transforming one of the schemata into the model of the other using best-effort translations and then perform intra-model matching and mapping generation between the two.

A variant of the first case makes use of a unified internal representation that encodes all relevant aspects from both the source and target model.

While all our previous examples have assumed direct mapping as a default, Figure 3.4 illustrates a simple *indirect* mapping case. The relational schema is first translated into a canonical ontology representation, which can then be aligned with the actual target ontology in a second step.

There are representative systems for both approaches in inter-model mapping, e.g., RONTO [23], which uses a variant of direct mapping, or BootOX [19], which follows the indirect approach.

Intuitively, the direct approach (or any variant of it) appears to be more appealing. First, it can naturally access native patterns. Second, at least when building a new system from scratch, there is no obvious reason for taking an architectural detour when it is also possible to proceed directly.

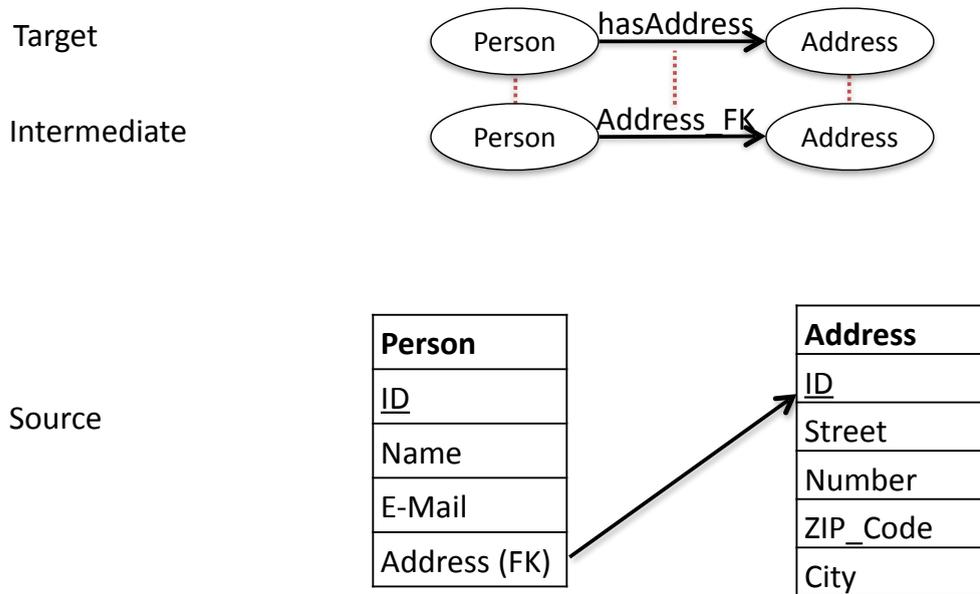


FIGURE 3.4: Simple inter-model matching using an indirect approach

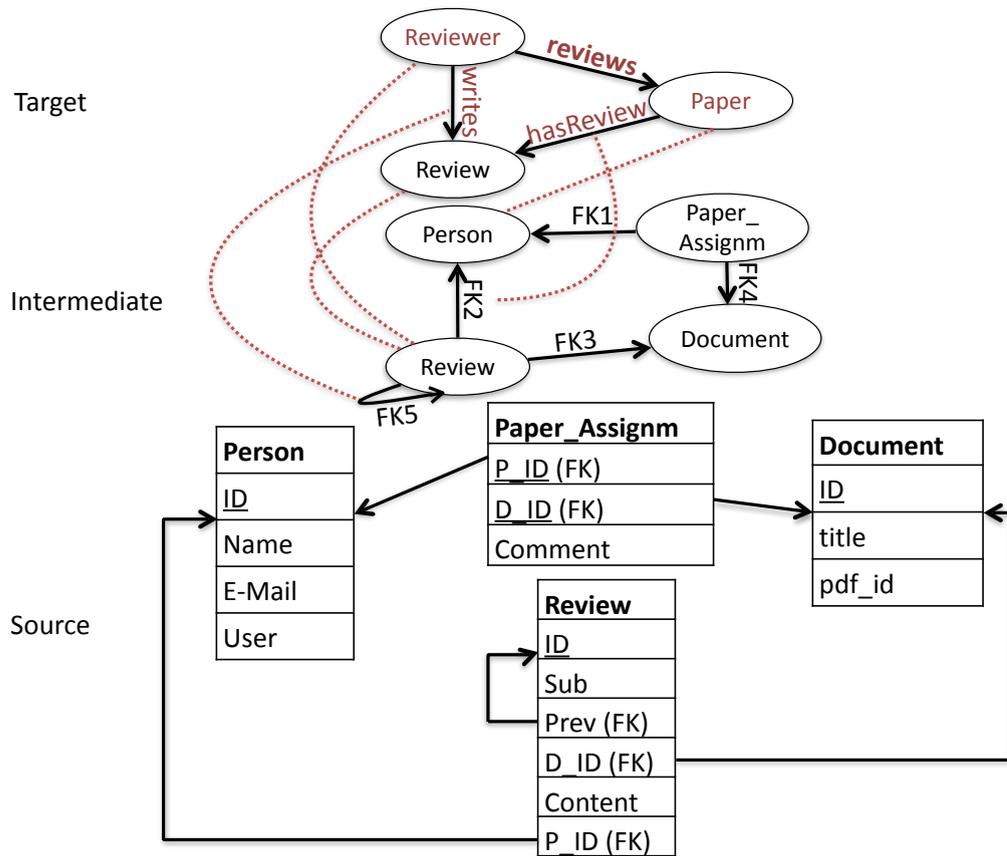
On the other side, there is a strong motivation for taking an indirect approach as well. Indirect mappings happen in two separate stages, so it is possible to re-use existing components in either one stage. Such components may already be available and could have been developed independently for producing intra-model mappings, i.e., for mappings between different schemata within the scope of the same data model. Existing components may bring a proven track record to the table and have the apparent advantage that their re-use reduces development efforts for any new mapping system.

At the same time, such reusable components have typically been developed for a different purpose, e.g., for generating *intra*-model mappings. Thus, they are not built to consider the specific challenges of any particular inter-model mapping.

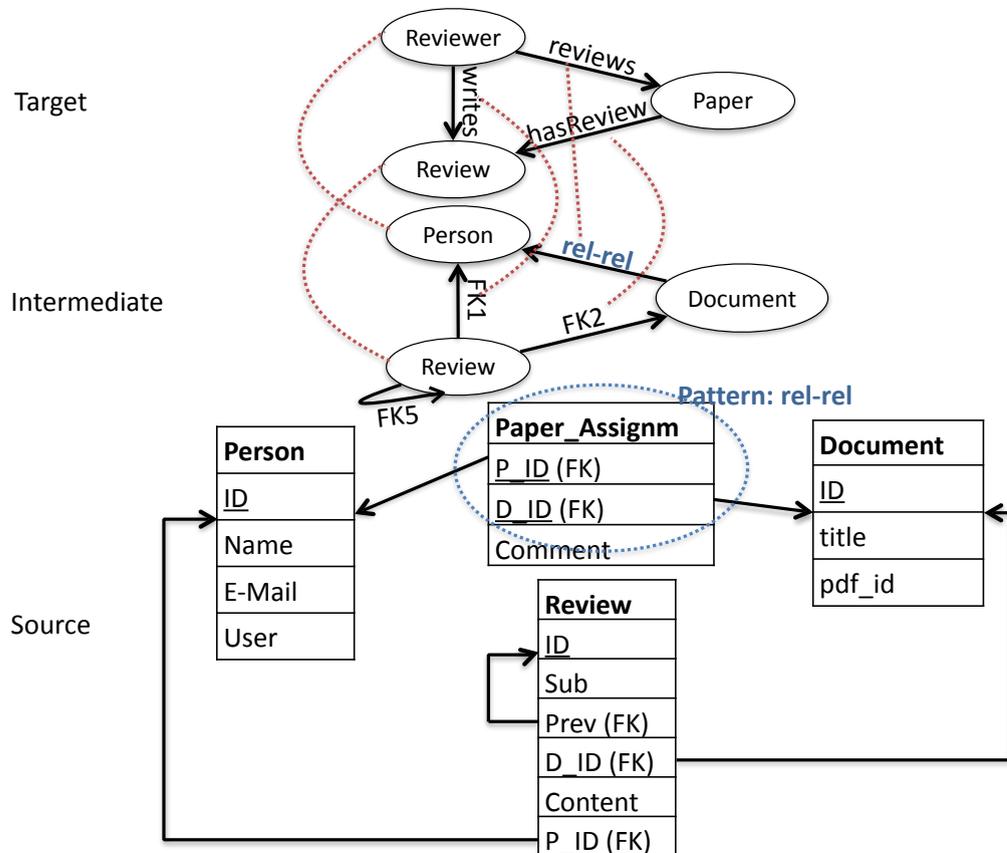
Figure 3.5 depicts indirect mapping scenarios for our example with papers, reviews and reviewers (Example 3.2).

First, Figure 3.5a illustrates the case of naive indirect mapping without patterns. This can easily lead to mismatches, as exemplified in the figure.

In a more ambitious approach some systems overcome parts of these limitations by considering patterns through a strategy of database reverse-engineering. Those systems carry their perception of patterns over to the intermediate ontology that they produce. Figure 3.5b shows how such systems could solve the problem just as well as a direct approach by translating the reverse-engineered pattern into the intermediate ontology.



(A) Indirect mapping without patterns (causing mismatches)



(B) Indirect mapping with basic pattern translation (correctly matched)

FIGURE 3.5: Inter-model matching scenario in an indirect matching approach with and without pattern translation through database reverse engineering

Still, applying reverse engineering this way means that a fixed and final semantic interpretation of each input pattern has to be decided before the actual mapping takes place. Consequently, those systems are bound to decide on exactly *one* interpretation of a pattern that they consider *globally* most likely. However, other interpretations may exist that could later become the most likely (and correct) interpretation given the mapping context. For instance, this would be the case if table *Paper\_Assignm* would not be a relationship relation after all, but would describe an entity that just happens to depend on two other entity types. This is a less frequent case but still happens. A typical example for this exception could be, e.g., a report, which is an entity but is defined by its topic (one external entity and table) and a type (which might be another entity, in a third table). A similar case could be constructed for the previous example if we consider a slightly more complex database schema, e.g., using the schema variant from previous Figure 3.3 with its ambiguous *4-way join* pattern.

One more alternative for using patterns in an indirect mapping approach would also be possible: converting the input model more or less naively into an intermediate ontology, but keep rich provenance as annotations. Provenance information could then be used by the eventual mapping generator to consider mapping patterns when aligning with the actual target ontology. However, to this end the mapping generator would need to be aware of provenance information, recognize relevant patterns, and consider them when generating mappings. This requirement somewhat counteracts the initial motivation to simply re-use existing mapping components, as components would neither be aware of additional provenance information nor could they act on the consequences of such information, e.g., by considering or preferring different mapping options. Also, to the best of our knowledge, no published systems use provenance this way for indirect RDB2RDF mapping generation.

Even though indirect matching approaches can be tuned and extended to overcome most of their limitations w.r.t. inter-model mapping generation as discussed above, this is not easily possible in every case. With *i<sup>3</sup>MAGE*'s IncMap we therefore follow the first (i.e., direct) approach to mapping generation.

### 3.1.2.5 Challenges in Incremental Matching

Next to mapping generation capabilities, *i<sup>3</sup>MAGE* also provides components for UI integration and effective interactions with human users. We adopt an interaction paradigm where a manual editing process is the default [131, 132]. Users can then either request specific mappings explicitly or, more frequently, make use of mapping *suggestions* that

are offered to them in context of their current actions, and which they could either accept or reject.

Despite these specific characteristics, incrementality plays a lesser role than inter-model patterns when it comes to specific challenges. This is because, to the most part, user feedback is solicited in a form that is similar to other, well-studied mapping generation approaches.

Two notable exceptions need to be considered:

1. Full incrementality: *i<sup>3</sup>MAGE* is designed to generate mappings incrementally, i.e., to produce partial mappings and also to accept hand-crafted or manually checked partial mappings as input to subsequent iterations. This is important because partial mappings can be seen as the most generic (and also most expressive) type of perceivable human input. Confirmed partial mappings also have another specific characteristic that separates them from most other forms of human input: they are final. On the one hand, they therefore are no longer useful to further refine the part of the mapping that they already describe themselves. On the other hand, they are only indirectly or weakly connected to other parts of a future mapping. The challenge is thus to still leverage those partial mappings to re-rank suggestions for other, indirectly related, parts of the mapping.
2. Interaction through query workload: *i<sup>3</sup>MAGE* is expected to proceed pay-as-you-go for mapping generation as needed. This means, most specifically, that a part of the mapping needs to be generated if data is to be queried but has not yet been mapped. As a consequence of this requirement, queries are often available as context for mapping. Therefore, a challenge is to leverage individual queries over the target schema as input for mapping generation.

Besides these two challenges, working interactively in a semi-automatic setup can be considered as a mere engineering challenge.

## 3.2 Matching and Mapping Generation Approach

At the core of *i<sup>3</sup>MAGE* is a mapping generator, IncMap [52], which has been built specifically to work in inter-model mapping tasks with further optimizations for RDB2RDF in particular. IncMap is also designed to scale between a fully automatic and a semi-automatic approach. The semi-automatic approach solicits human feedback but also leverages manually curated partial mappings to adjust the automatic mapping and to

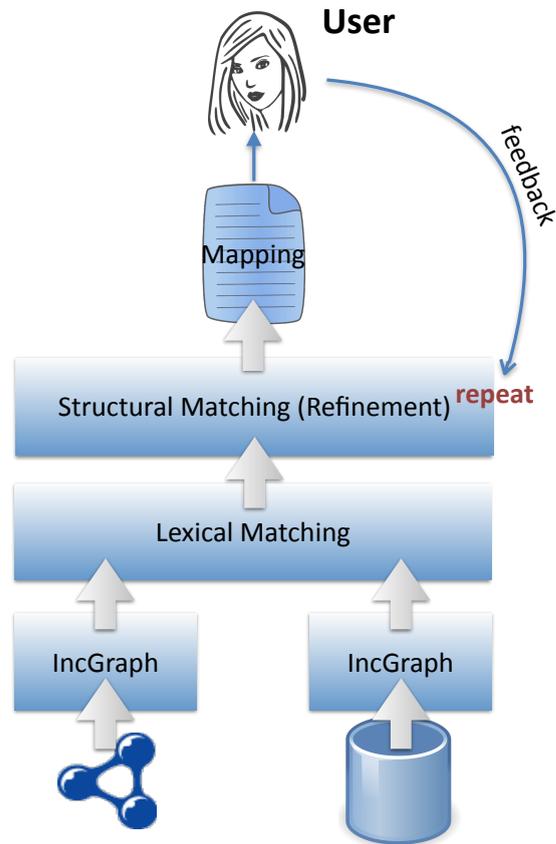


FIGURE 3.6: High-level view of mapping process with i<sup>3</sup>MAGE

re-rank suggestions. In addition, it can use a query workload as context to locate areas of interest in the target schema and identify the most relevant semantic connections.

Figure 3.6 depicts an overview of the mapping generation process in i<sup>3</sup>MAGE with IncMap. The system automatically generates RDB $\mathcal{2}$ RDF mappings based on iterative schema graph matching and operates in five stages:

1. Creating source and target schema graphs (IncGraph)
2. Reasoning and heuristic pattern annotation to infer additional information
3. Initial lexical matching to build a matching graph
4. Refining matching scored using a fixpoint computation
5. Mapping generation

In the following, we give an overview of these steps. Details and a more formal explanation are given afterwards in Chapter 4.

### 3.2.1 Creating Schema Graphs

As a first step we need to build schema graphs for both source and target. Those can later serve as input to the matching computation. We have devised a dedicated data structure, IncGraph, to represent relational schemata and ontologies in a unified, yet model-aware fashion as schema graphs. To construct the IncGraphs, we iterate over all elements in the input schemata (i.e., axioms such as class definitions and relational schema elements such as table declarations).

**Example 3.3** (Papers and Authors). *Consider a small ontology, again in the conference domain, which simply models authors who write papers. The ontology consists of concepts Author and Paper, some datatype properties of those concepts and an object property that connects them (writes). To simplify, papers in this example have only one author. Figure 3.7 depicts this scenario with a relational schema that captures the information (Figure 3.7a), an ontology (Figure 3.7b) and IncGraphs for both of them (Figures 3.7c and 3.7d).*

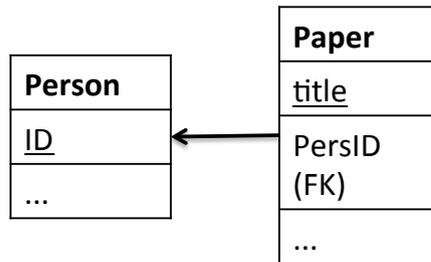
The relational schema and the ontology in this example both capture the same information.<sup>2</sup> However, while *Author* is a dedicated concept in the ontology, in the relational schema this classification is established only implicitly by *Persons* who have authored *Papers*. The resulting *basic* IncGraph representations in Figure 3.7c and Figure 3.7d already show a number of potential direct correspondences: Classes may correspond to database tables, datatype properties may correspond to attribute values, object properties correspond to referential constraints, and explicit datatype ranges provide means for identification.

The relations between nodes are generalized to basic roles such as referencing, acting as the smallest common denominator between both models. Moreover, on the database side referentially constrained attributes (i.e., foreign keys) are modeled both as values and references to allow them to be matched in both their capacities. Also, edges representing referential constraints are effectively undirected.<sup>3</sup> This is because referential constraints have no *semantic* direction, whereas the semantic direction present in object properties does not yield cardinality information (which is implied by a foreign key's direction).

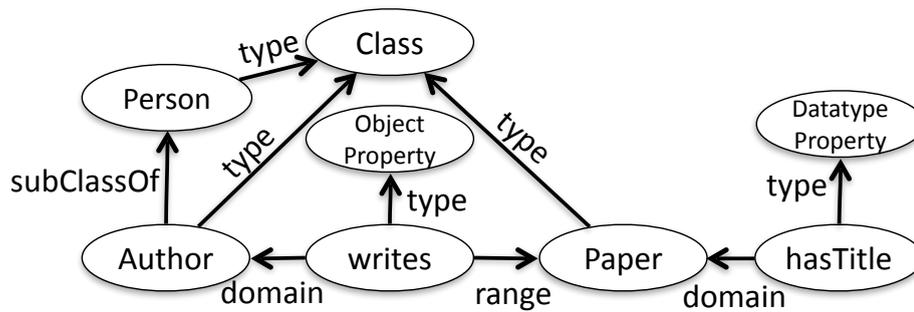
Note, that we use different colors of nodes for the different aspects. In Figures 3.7ff, class and table nodes are white, nodes of properties and referential constraints are light blue, type nodes are green, etc. When matches are calculated in the next step, only

<sup>2</sup>Apparently, the depicted ontology is semantically richer. Both, however, can be considered a schema to accommodate the exact same data or A-Box facts.

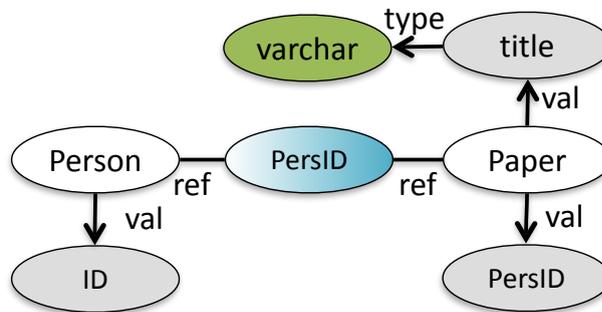
<sup>3</sup>Technically, there are two edges in opposite directions, having a combined effect that is equivalent to undirected edges w.r.t. our algorithms.



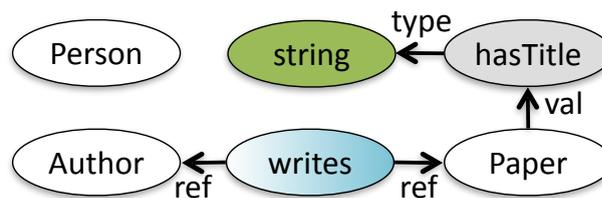
(A) Relational schema  $R$



(B) Ontology  $O$



(c) Basic IncGraph( $R$ ) (simplified)



(D) Basic IncGraph( $O$ ) (simplified)

FIGURE 3.7: Relational schema and ontology with corresponding basic IncGraph representations

nodes of the same color need to be considered as correspondences. For instance, classes will only be matched with tables.

### 3.2.2 Reasoning and Patterns

Figure 3.7 also highlights one of several possible distortions between the IncGraph representation of relational schemata and ontologies: intuitively, the *Person* class would be the most accurate match for the *Person* table, however, the node is disconnected from the rest of the graph making it a structurally less attractive match. This is because the *subClassOf* connection between *Author* and *Person* is not modeled as it cannot have a correspondence in the relational database.

For this reason, as a second step we apply reasoning techniques on the input ontology and use heuristics to annotate modeling patterns on the source database. Figure 3.8 depicts knowledge derived from reasoning on the ontology and also adds detected patterns to the IncGraphs from Example 3.3.

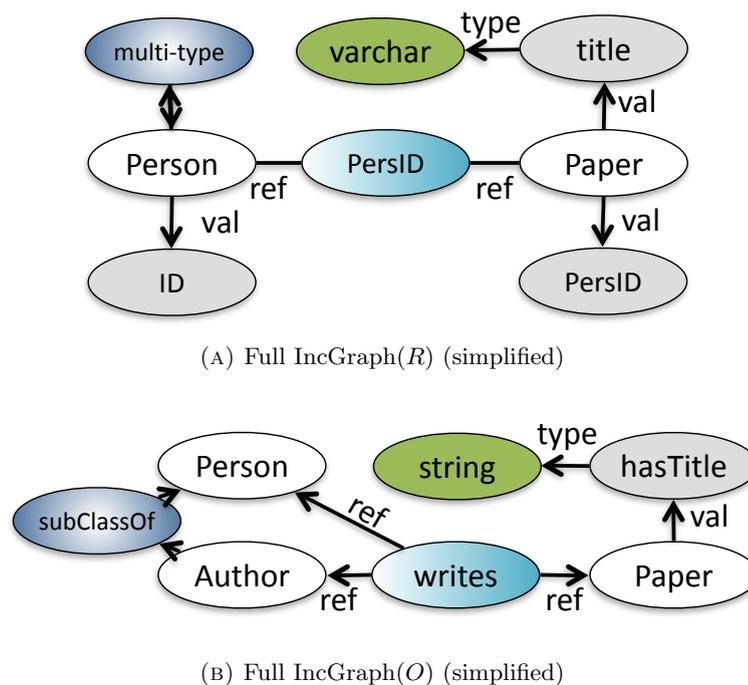


FIGURE 3.8: Schema and ontology with advanced IncGraphs

In Figure 3.8a, the relational IncGraph now annotates the *Person* node with a pattern, which *heuristically* states that this table very likely contain *individuals* of several types (e.g., using subclasses or sibling classes). At the same time the ontology IncGraph in Figure 3.8b now contains a dedicated node for the *subClassOf* axiom. This information can now be used to derive a new correspondences with the relational side.

Additionally, a reference edge is added to the ontology graph, which directly connects *Person* and *Writes*. This knowledge is derived through reasoning and basically states that some persons write papers. In our example this encourages correspondences along the path of  $(Person/Person) - (writes/PersID) - (Paper/Paper)$ , which would be the most accurate alignment of the input schemata. The additional edge, however, is down-weighted to cover for the fact that only *some* persons would actually write papers, making it a less likely correspondence in the general case.

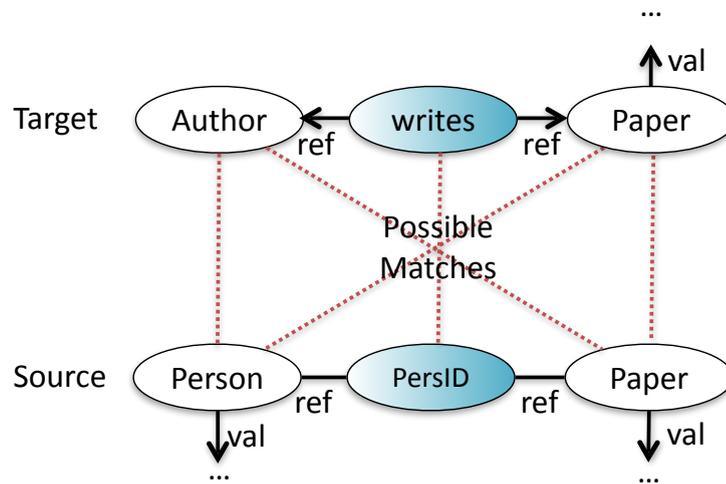
### 3.2.3 Matching

Based on a source and target schema IncGraph, we next calculate the initial matching graph. Figure 3.9 illustrates a simplified match of the basic IncGraphs from Example 3.3. For every color (or type) of nodes the cross product is matched into paired nodes. Possible matches are shown in Figure 3.9a: all concepts could match with any table (cross product), but for node *writes* there is only one possible match as there is only a single node of corresponding color in the source graph. Paired nodes are then connected if their inner nodes (source and target) were each connected with the same edge type in the original IncGraph. All paired nodes also receive an initial score using lexical matching. Intuitively, several alternative alignments form sub graphs, as shown in Figure 3.9b.

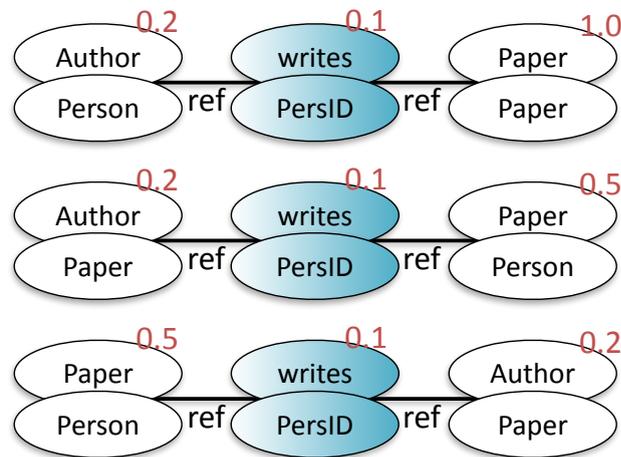
### 3.2.4 Fixpoint Computation

In subsequent next steps, IncMap calculates a modified matching graph, preparing it to perform a fixpoint computation. Matching graph transformation closely follows the process described in [45] for Similarity Flooding. Changes to the initial matching graph aim to balance the influence of edges. The fixpoint computation serves to refine match scores based on the graph structure. Intuitively, the process favors nodes in larger sub graphs over smaller ones and increases the scores of strongly connected nodes. We introduce Similarity Flooding in more detail in Chapter 4.

Different from the original Similarity Flooding algorithm, we introduce modifications for features such as weighted edges in the input graphs and selectively activating edges during the fixpoint computation (e.g., based on patterns). Moreover, in order to additionally support incremental mapping scenarios where user feedback is available, we can accommodate information on partial mappings. Despite its incremental, interactive process, IncMap is also able to generate mappings completely without any user feedback.



(A) Source and Target IncGraphs



(B) Matching Graph

FIGURE 3.9: Matching IncGraphs (simplified)

### 3.2.5 Mapping Generation

Finally, mappings are generated from the correspondences in the matching graph.

In the fully automatic mapping case, a selection is made from the set of all correspondences. From the initial Cartesian matching of colored nodes, relevant correspondences can be grouped in four categories: class-table correspondences, property-reference correspondences, property-attribute correspondences, and pattern-supported correspondences. Due to the way that IncGraphs are constructed, properties always appear in the context of one concrete domain and range (i.e., they can occur several times for different specific domain/range pairs). Thus, each correspondence can be translated straightforward into a simple mapping rule. Correspondences are selected based on their similarity score of the fixpoint computation and consistent alignment interpretation. Consistency in this

context means that we will, for each class, select only at most one corresponding table and choose property correspondences only where domain and range interpretations are consistent with previously chosen classes.

For interactive mappings with suggestions, a suggestion selector reads and presents a specific selection of correspondences in context (e.g., all *top-3* matches that are somehow associated with a specific class or table). In this case, no automatic selection of correspondences takes place. Instead, the suggestion selector requests mapping generation for individual confirmed matches, only.

To enable mapping generation technically, we encode provenance information with all nodes.

### 3.2.6 Incremental and Interactive Mapping Generation

User feedback on individual correspondences or partial corrected mappings can be used to refine suggestions iteratively. In this case the fixpoint computation and mapping generation will be repeated after each round of feedback.

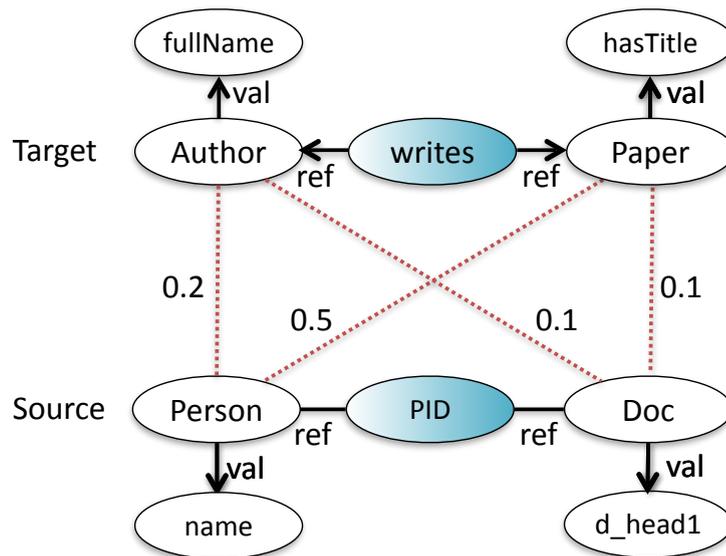
#### 3.2.6.1 Incremental Mapping

Incremental mapping works iteratively with partial mappings at different stages of completions [133]. This process allows us to leverage user feedback after each iteration to improve the quality of mapping suggestions in subsequent iterations. One of the reasons why we have chosen Similarity Flooding [45] as a basis for *i<sup>3</sup>MAGE*'s IncMap is the fact that user feedback can be easily integrated by adopting the initial match scores in a graph before the fix-point computation starts.

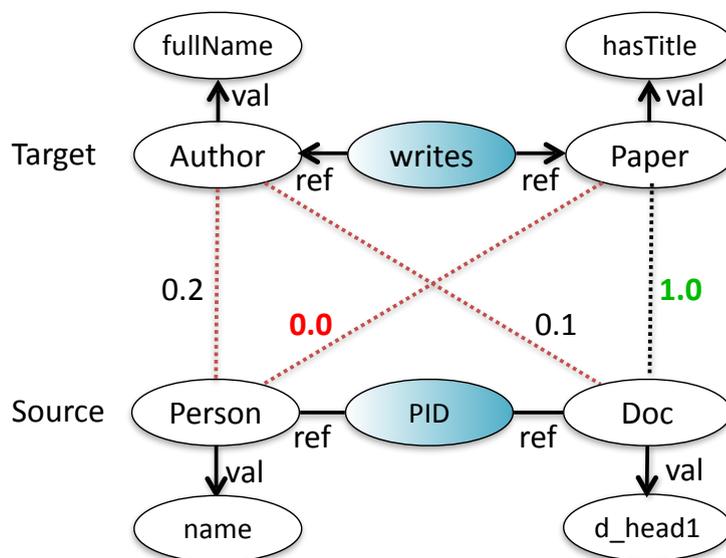
Although the possibility of an incremental approach has been mentioned already in the Similarity Flooding paper, it so far has not been implemented and systematically evaluated. Also, while it is simple to see *where* user feedback could be incorporated, it is far less trivial to decide *which* feedback should be employed and *how* exactly it should be integrated in the graph.

Figure 3.10 illustrates the general principle of including feedback that is usable in Similarity Flooding. Initially, matches yield scores that have been assigned by some previous processing step, as depicted in Figure 3.10a (only a subset of correspondences is actually shown in the figure).

After a round of feedback, while most nodes still yield the initial matching score calculated in the previous phase, some are confirmed or rejected matches. They yield a score



(A) Source and target with selected correspondences



(B) Correspondences after confirmations (green) and rejections (red) of correspondences for “Paper”

FIGURE 3.10: Incorporation of feedback

of 1.0 (confirmed) or 0.0 (rejected). In Figure 3.10b, they are printed in green and red, respectively.

Feedback can be given in form of an explicit user interaction or in form of partial mappings. We focus on leveraging only the most important and most decisive kind of user feedback, i.e., the previous confirmation or rejection of suggested mappings. We have devised and tested three alternative methods how to add this kind of feedback into the graph.

The main one follows a largely straightforward idea: it simply maximizes the influence of feedback from the modification throughout the fix-point computation. Instead of just initializing a confirmed or rejected match with their final score once, we repeat the initialization at the end of each step of the fix-point computation after normalization. This way, nodes with definite user feedback influence their neighborhood with their full score during each step of the computation, while changes from nodes with non-definite scores do not affect them in return. We therefore call this method *Self-Confidence Nodes*. We introduce and compare all implemented methods in detail in Chapter 4.

Incremental mapping as a process can also be stirred by queries that implement certain information needs and are provided pay-as-you-go. If this is the case, then those queries can also be used as input for matching. We leverage known information needs to locate areas of current interest in the target schema and increase the score for all matches related to schema elements in the scope of current interest. This introduces a bias for match constellations that consistently cover all of the area of interest. In addition, they highlight specific sub-type interpretations as part of this bias, which may hint to more realistic matches. For instance, if a query asks for authors who know other authors, and assuming that this query can be reasonably answered over the available data, then it is more than likely that such a connection is explicitly modeled in the underlying relational database. At the same time, no connection between two persons might be explicitly modeled in the database, even if the domain and range of *knows* in the ontology is *Person*, rather than *Author*.

### 3.2.6.2 Interactive Feedback

Apparently, incremental mappings require some form of external input to control, which parts of the mapping are to be calculated at which point in time. This external input usually comes in the shape of human interactions. Essentially, a human user chooses parts of the source or target schema that are of immediate interest to them and should thus be mapped.

Due to the interactive nature of controlling such incremental steps, this opens a natural opportunity for additional human feedback on each such calculated increment.

In our approach increments can be triggered from various contexts, e.g., when a user manually edits a mapping or when a user query taps into still-unmapped parts of the A-Box. We then offer atomic mapping suggestions based on a single match, which can be accepted or rejected. Mapping suggestions from IncMap can be transformed into mappings in a most straightforward manner (c.f. Section 3.2.5), and thus directly relate back to its underlying match.

If a user accepts or rejects a suggestion we thus confirm or obliterate the related match in the matching graph and re-initiate the fixpoint computation to update related suggestions.

A more difficult case arises from increments that have been performed *outside* the control of *i<sup>3</sup>MAGE*, i.e., when partial mappings have been added fully manually. In this case we check those mappings to derive certain IncMap matches that they corroborate, and confirm those. We cannot, however, obliterate any matches. Also, our grip on understanding the semantics of manually curated mappings is limited in some cases for technical reasons.

## Chapter 4

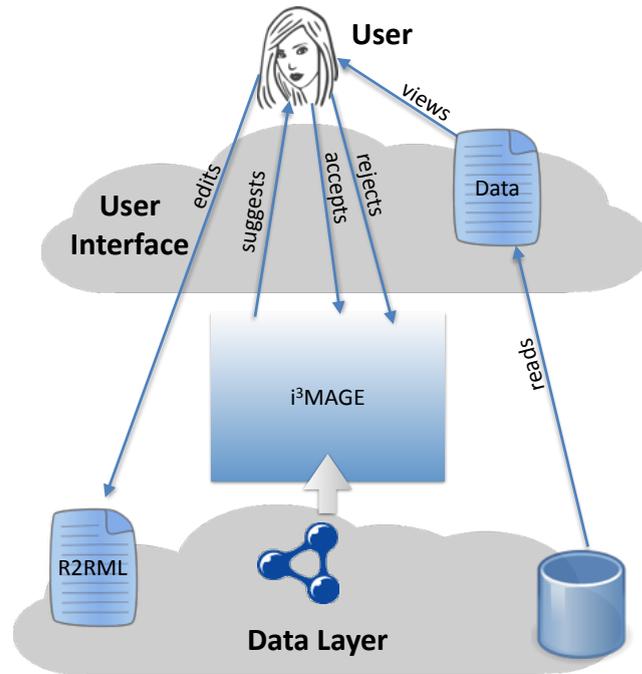
# i<sup>3</sup>MAGE System

In this chapter, we give a detailed description of the i<sup>3</sup>MAGE system. We first give an architectural overview in Section 4.1, before we discuss the details of the IncGraph model used by i<sup>3</sup>MAGE’s core component IncMap in Section 4.2. In Section 4.3, we add additional background on the Similarity Flooding algorithm used by IncMap. Then, we discuss the implementation of matching in Section 4.4. In Section 4.5, we present the implementation of correspondence selection and mapping generation in i<sup>3</sup>MAGE. Finally, we describe additional components of i<sup>3</sup>MAGE in Section 4.6.

### 4.1 System Overview and Architecture

From a high-level point of view, i<sup>3</sup>MAGE can be seen as a system that primarily provides mapping suggestions, translates them into actual mappings upon request and refines its suggestions from the feedback harvested during this process. Figure 4.1 illustrates this high-level perspective from an end-user’s viewpoint.

Users communicate with a user interface (UI) that allows them to somehow view underlying data or interact with it. Depending on the background of the users and the system, they could possibly even modify the mappings that make such data accessible. With i<sup>3</sup>MAGE in the picture, the UI also provides suggestions on how semantic bits of data could be further mapped, i.e., how data can be enriched with individuals. Those suggestions can appear inside a mapping editor or in any other place where the user-facing system realizes that additional data need to be mapped. Users interact with suggestions by accepting or rejecting them. They can watch the consequences of their decision in changes to the data that they view. This behavior is made possible by a multi-component architecture.

FIGURE 4.1: *i<sup>3</sup>MAGE*: High-level, end-user's perspective

#### 4.1.1 System Architecture

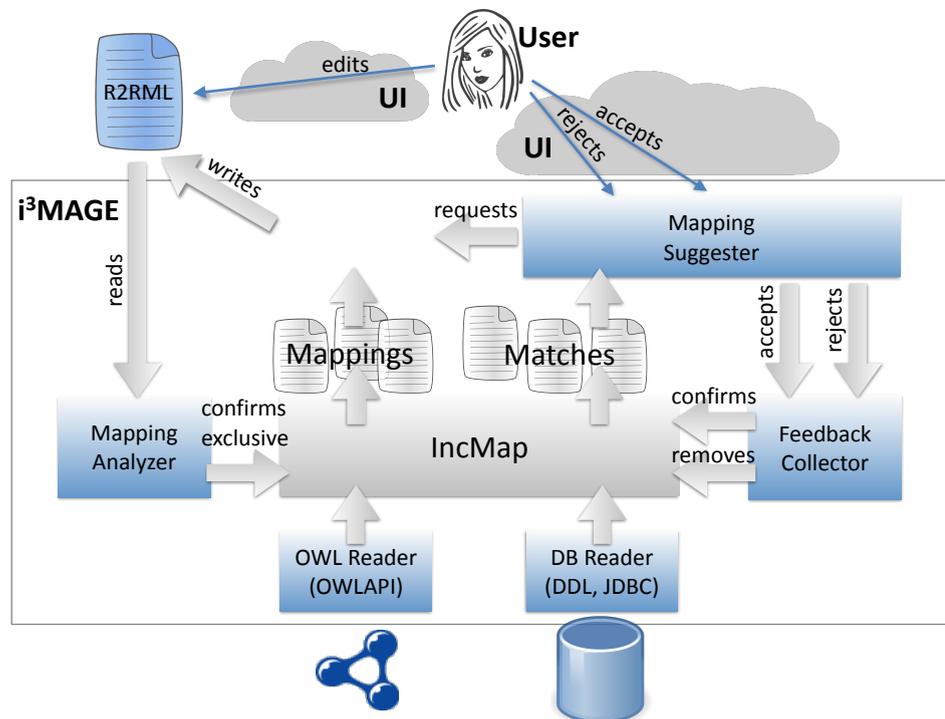
FIGURE 4.2: Overall architecture of *i<sup>3</sup>MAGE*

Figure 4.2 depicts the overall architecture of *i<sup>3</sup>MAGE*. At its heart is IncMap [52], our incremental matching and mapping generation system. Looking at the architecture

diagram from bottom right clockwise, further components of *i<sup>3</sup>MAGE* are:

- A database reader that connects to running DBMSs, reads their schema information and transforms those information for IncMap’s API. The database reader also accepts textual DDL as an alternative.
- An OWL ontology reader that loads mapping target ontologies using the OWL API [134] and transforms relevant axioms for IncMap’s API.
- A mapping analyzer, which reads existing partial R2RML mappings, derives correspondences from them and feeds those to IncMap. Partial mappings could be the result of previous iterations running *i<sup>3</sup>MAGE*, or they could have been manually curated.
- A mapping writer, which reads IncMap’s mappings and exports them as plain R2RML mappings. The mapping writer exposes mappings using an API, but can also serialize them in Turtle RDF format, as required by the R2RML standard [77].
- A mapping suggester that offers an API to find and pick suggestions by context (e.g., related to one particular class or table). The mapping suggester also translates suggestions into human readable form, including some explanation.
- A UI is not part of *i<sup>3</sup>MAGE*, but required for users to interact with the system and thus closely connected.
- A feedback collector accepts feedback on suggested mappings indirectly via the mapping suggester, and forwards it to IncMap.

Figure 4.2 illustrates the inner components and architecture of IncMap. From bottom to top, IncMap exposes an API for setting schema elements of the source and target schemata through a number of setter methods (*Schema API*). This native schema information can then be used by IncGraph builders to construct an IncGraph from either relational schema information (*DDL IncGraph Builder*) or from an OWL ontology (*OWL IncGraph Builder*).

Next, the relational source IncGraph and OWL target IncGraph need to be connected. To this end, an *extended pairwise connectivity graph* (PCG+) will be built by the *PCG+ Generator*. A PCG+ is based on a graph structure that has been first introduced by Melnik et al. [45], which we have extended for our purposes in IncMap. Basically, the idea of the PCG+ is to calculate a Cartesian product from all valid combinations of nodes in the two IncGraphs and then to reconstruct shared edges between the resulting nodes.

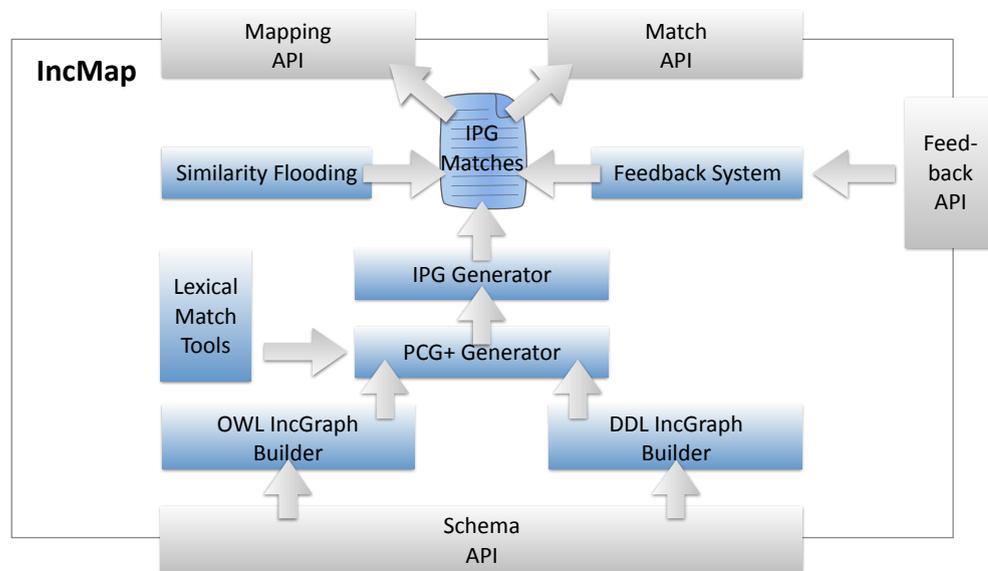


FIGURE 4.3: IncMap architecture

The PCG+ generator also assigns initial scores to all paired nodes in the PCG+ using one of the built-in *Lexical Match Tools*.

Next, the *IPG Generator* takes a PCG+ graph and transforms it into an *induced propagation graph*. The IPG is a variant of the PCG+ that can be used for running a Similarity Flooding fixpoint computation. PCG+ generation as well as IPG generation closely follow the process laid out in the Similarity Flooding paper [45]. We discuss Similarity Flooding in more detail in Section 4.3.

As an artifact from all of the previous steps combined, the IPG graph is the central asset during incremental and interactive mapping of any two schemata. It already encodes all possible matches, with their initial scores. Raw matches as well as materializable mappings can be generated from it at any point. To refine the matches based on graph structure, the *Similarity Flooding* component can be executed with an IPG as input, and will produce a refined IPG as output. Similarly, the *Feedback System* will adjust the IPG over time whenever it learns about the correctness of individual matches in the graph.

#### 4.1.2 Implementation Notes on Architecture

All of the architecture of *i*<sup>3</sup>MAGE described above is implemented in Java. The different components expose APIs to allow interaction with their peer components. The system

requires a dynamic configuration for connecting with databases, ontologies, etc. Mapping suggestions, mapping export, and feedback are again handled through APIs exposed to external components, e.g., a user-facing system. *i<sup>3</sup>MAGE* is thus designed to work as a technically independent component, running in-process of other applications written in Java or compatible languages. Naturally, such an application could also take the shape of a Java wrapper that exposes the functionality of *i<sup>3</sup>MAGE* as a server to connected clients.

## 4.2 IncGraph

In this section, we describe the IncGraph model used by IncMap to represent schema elements of an OWL ontology  $O$  and a relational schema  $R$  in a unified way. The IncGraph model is defined as a directed labeled graph, which can be used as input for matching.

As the structural matching process follows along the lines of Similarity Flooding [45], IncGraphs are designed to be a suitable input for further processing with Similarity Flooding. This idea is not new in principle. Most prominently, COMA++ [24] is based on the same concept for generic, graph based inter-model matching. IncGraph stands out by optimizing the graph structure specifically to reduce the inter-model gap by assuming a unifying basic structure and by applying a number of further unifying annotations, as we discuss below.

### 4.2.1 Running Example

We will use the following running example to explain the various steps of IncGraph construction.

**Example 4.1** (Papers and Authors). *Consider an ontology in the conference domain, which models authors who write papers. The ontology consists of concepts Author and Paper, some datatype properties of those concepts and an object property that connects them (writes). Figure 4.4 depicts this scenario with the target ontology (Figure 4.4a) and a relational schema that captures the same information (Figure 4.4b).*

Different variations of IncGraphs can be constructed for both the relational and ontology sides, as we will illustrate in the following.

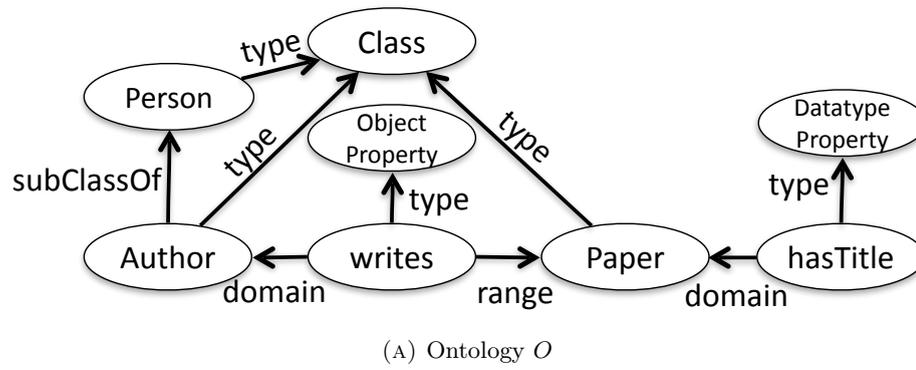
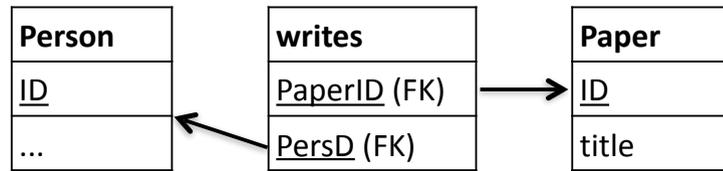
(A) Ontology  $O$ (B) Relational schema  $R$ 

FIGURE 4.4: Running example for IncGraph construction (input target ontology and relational source schema)

### 4.2.2 Modeling Assumptions

**Ontology Schema  $O \in \mathcal{O}$ :** As ontology schemata (c.f. Definitions 2.3) we denote all schematic elements (c.f. Definition 2.1) from an ontology, usually the axioms in its T-Box. We consider OWL 2 ontologies [29]. For all reasoning-related considerations, we assume RDF-based semantics.

For all ontologies, we assume and require a clear majority of classes and properties to be declared with an explicit IRI using an OWL class expression.<sup>1</sup> In addition, we assume that a majority of properties yields precise domain and range information, i.e., with a domain other than *Thing* and a range other than *Thing* or *PlainLiteral*.

**Relational Schema  $R \in \mathcal{R}$ :** A relational schema  $R$  (c.f. Definition 2.4) is described by a set of relational schema elements (c.f. Definition 2.1). Each relation (table) is defined by its unique label and is associated with the set of its attributes (columns), which are identified by their labels and may have additional information restricting the attribute domain (datatype). Subsets of attributes can be constrained to hold unique values (i.e., to form a candidate key), and we require at least one such key to be explicitly defined in each relation as primary key. Additionally, we consider referential constraints (foreign keys) to define references between different relations.

<sup>1</sup>While blank nodes are not a hindrance for building IncGraphs, our matching partially relies on the presence of IRIs.

This definition is slightly relaxed compared to the original definition of the relational data model [27] or of the common modern SQL variants [28, Chap. 2]. While the former usually also require some kind of order of attributes or make attribute domains a mandatory part of attribute definitions, we leave out any notion of attribute order and consider attribute domains as optional constraints. Common definitions of relational models can thus deterministically project to our definition, although the opposite direction is not necessarily possible.

### 4.2.3 IncGraph Model Definition

The main goal of the IncGraph model is to represent the schema elements of ontology  $O$  and relational schema  $R$  in a unified way, and make it a suitable input for graph matching.

An IncGraph model is defined as a labeled graph  $G = (V, lbl_V, E, lbl_E, W_E)$ . It can be used as input by the matching algorithm of IncMap.

$V$  is a set of colored vertices. Each vertex (node) in the graph can be of one of the following colors (types):

- object/table ( $T$ ): represents schema elements that can have class characteristics, e.g., ontology classes or tables. Depicted *plain white* in the figures.
- reference ( $R$ ): represents a relation between objects, e.g., an OWL object property or a relation supported by a foreign key constraint. Depicted in a *light blue shade*.
- data attribute ( $A$ ): represents elements in the schema that act as handles for atomic literal data, e.g., datatype properties or individual attributes. Depicted *light grey* in the figures.
- datatype ( $D$ ): value domains of data nodes, e.g., the range of a datatype property or the type of an attribute. Depicted in the figures in *green*.
- pattern type ( $X$ ): in advanced IncGraphs only, meta information about knowledge observed in the original input schemata can be encoded and introduce specific features of the input model for matching in a unified way. Patterns can be the canonical representation of ontological features such as a subclass relationship, or could describe relational model patterns, such as relationship relations. Depicted *dark blue* in the figures.
- pattern ( $Y$ ): each occurrence of a pattern is modeled by a dedicated pattern node. Depicted as a *dark blue radiant with a lighter center*.

$E$  is a set of edges. There are labeling relations  $lbl_V$  and  $lbl_E$  for vertices and edges, respectively. They relate exactly one label to each vertex or edge in the graph.  $W_E \subset E \times [-m; m]$  is a weight assignment relation for edges, where  $m$  denotes the maximum weight.<sup>2</sup>

In our implementation, we apply a maximum edge weight of  $m = 2.0$ . The default edge weight is 1.0. For brevity, we assume in the following that edge weight is 1.0 for all edges, unless explicitly stated otherwise.

Label  $l_v$  is the label of  $v \in V$  if  $(v, l_v) \in lbl_V$ , and represents a name of a schema element. Similarly,  $l_e \in \{“ref”, “val”, “type”, “pe”, “pi”, “pt”\}$  is a label of edge  $e \in E$  if  $(e, l_e) \in lbl_E$  and describes the function of the edge as follows:

- *ref*: reference, as in object properties or referential constraints.
- *val*: leads to a data node, as in datatype properties or attributes.
- *type*: point to the datatype of data values (e.g., XSD types or SQL types).
- *pe*: pattern end-point.
- *pi*: pattern inner node.
- *pt*: points to a pattern type.

The apparent redundancy between the types of edge and the nodes that edges are pointing to is a necessary feature to effectively and efficiently process IncGraphs later on for matching: when graphs are paired, edge labels act as a filter to consider only combinations of connections that can potentially have the same semantics.

#### 4.2.4 Basic IncGraph Construction

Let  $R \in \mathcal{R}$  be a relational schema,  $O \in \mathcal{O}$  an ontology.

Basic nodes (vertices) and edges for IncGraph are based on input schema elements, i.e., tables and attributes for IncGraph( $R$ ) or classes and properties for IncGraph( $O$ ).

##### 4.2.4.1 Relational Schemata (IncGraph( $R$ ))

Let  $T$  the set of tables (relations) in the schema,  $A_t$  the set of attribute of table  $t \in T$ ,  $P \subset \{(t_1, a_1, t_2, a_2) | t_1, t_2 \in T, a_1 \in A_{t_1}, a_2 \in A_{t_2}\}$  the set of non-compound referential constraints between tables in  $R$ . Then:

<sup>2</sup>Negative edge weights are a mere technicality. We denote optional edges with negative weights. They can then be activated during matching by multiplication of their weight with  $-1.0$ .

**Table Nodes:**

$$t \in T \rightarrow v_t \in V \wedge v_t.type = "T" \wedge (v_t, name(t)) \in lbl_V$$

**Attribute Nodes:**

$$a \in A_t \wedge t \in T \rightarrow v_a \in V \wedge v_a.type = "A"$$

$$\wedge (v_a, name(a)) \in lbl_V \wedge e_a = (v_t, v_a) \in E \wedge (e_a, "val") \in lbl_E$$

**Datatype Nodes:**

$$v_a \in V \wedge v_a.type = "A" \rightarrow v_{dt} \in V \wedge v_{dt}.type = "D"$$

$$\wedge (v_{dt}, name(dt(v_a))) \in lbl_V$$

$$\wedge e_{dt} = (v_a, v_{dt}) \in E \wedge (e_{dt}, "type") \in lbl_E$$

**Reference Nodes:**

$$(t_1, a_1, t_2, a_2) \in P \rightarrow v_p \in V \wedge v_p.type = "R"$$

$$\wedge (v_p, name(a_1)) \in lbl_V \wedge e_{p_1} = (v_{t_1}, v_p) \in E$$

$$\wedge (e_{p_1}, "ref") \in lbl_E \wedge e_{p_2} = (v_{t_2}, v_p) \in E$$

$$\wedge (e_{p_2}, "ref") \in lbl_E$$

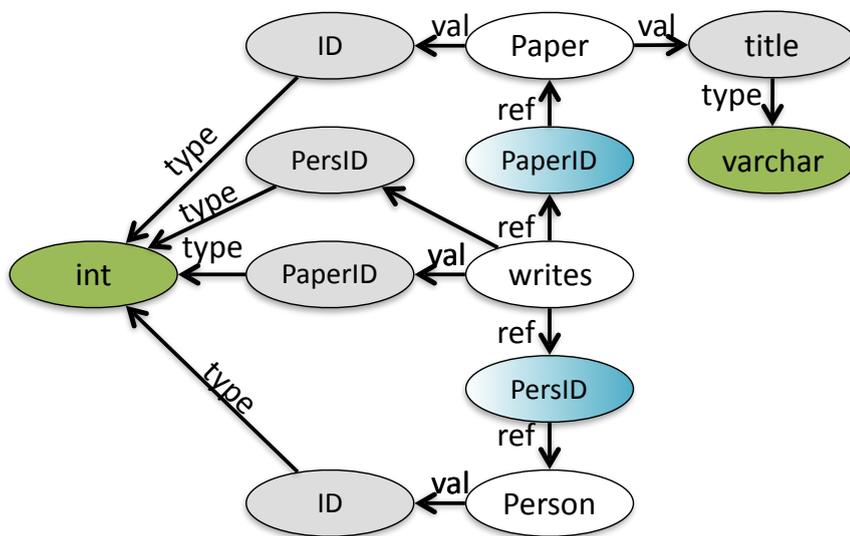
FIGURE 4.5: Basic IncGraph( $R$ )

Figure 4.5 shows a basic IncGraph constructed from running Example 4.1 for the relational schema according to the above definition. Edge weight is 1.0 for all edges, as none of the above rules specifies a deviation from default.

#### 4.2.4.2 Ontologies (IncGraph(*O*))

Let  $C$ ,  $D_P$ ,  $O_P$  the set of class axioms, datatype property axioms, and object property axioms in  $O$ , respectively, and  $X$  be the set of OWL datatypes. Then:

##### Class Nodes:

$$c \in C \rightarrow v_c \in V \wedge v_c.type = "T" \wedge (v_c, name(c)) \in lbl_V$$

##### Datatype Property Nodes:

$$\begin{aligned} d \in D \wedge c = domain(d) \in D_P \\ \rightarrow v_d \in V \wedge v_d.type = "A" \wedge (v_d, name(d)) \in lbl_V \\ \wedge e_d = (v_c, v_d) \in E \wedge (e_d, "val") \in lbl_E \end{aligned}$$

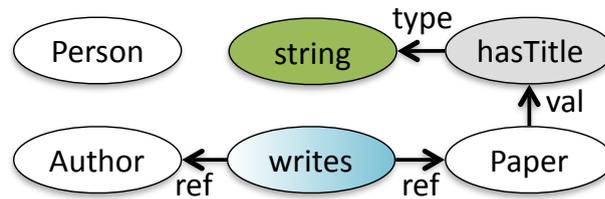
##### Datatype Range Nodes:

$$\begin{aligned} v_d \in V \wedge v_d.type = "A" \wedge r = range(v_d) \in X \\ \rightarrow v_x \in V \wedge v_x.type = "D" \wedge (v_x, name(r)) \in lbl_V \\ \wedge e_x = (v_d, v_x) \in E \wedge (e_x, "type") \in lbl_E \end{aligned}$$

##### Object Property Nodes:

$$\begin{aligned} p \in P \wedge d = domain(p) \wedge r = range(p) \\ \rightarrow v_p \in V \wedge v_p.type = "R" \wedge (v_p, name(p)) \in lbl_V \\ \wedge e_{p_1} = (v_d, v_p) \in E \wedge (e_{p_1}, "ref") \in lbl_E \\ \wedge e_{p_2} = (v_r, v_p) \in E \wedge (e_{p_2}, "ref") \in lbl_E \end{aligned}$$

Figure 4.6 shows a basic IncGraph constructed for the ontology from running Example 4.1. Again, edge weight is 1.0 for all edges, as none of the rules for the basic definition specifies any deviation from the default.

FIGURE 4.6: Basic IncGraph( $O$ )

### 4.2.5 Unifying Annotations

We extend IncGraphs with further nodes and edges, which we call annotations. Different types of annotations add further implicit schema information to the basic IncGraphs during the construction phase. The goal of these annotations is to further unify the structure of the IncGraph models with respect to different design principles that are applied during the modeling of ontologies and relational schemata.

Adding annotations must be done carefully since additional graph elements might also blur the schema structure, i.e., add noise. This can have a negative effect on the quality of the results of the matching process used by IncMap.

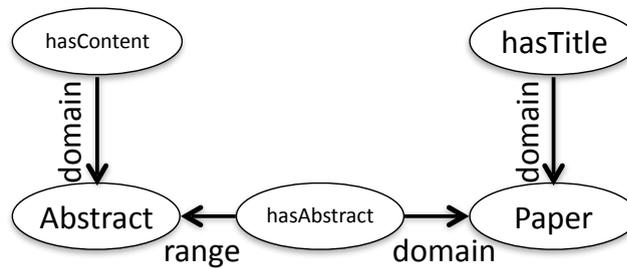
Therefore, we add some annotated edges in an inactive state to the IncGraph models (i.e., such that they are initially not used for matching). IncMap can selectively activate inactive edges during the matching process. They are encoded as having a negative edge weight, and IncMap implements a special handling for edges with negative weight. There is no need for any deactivation of annotated nodes, as nodes that are only (or mostly) connected through inactive edges behave the same as nodes that are disconnected (or poorly connected). Structural matching will assert that disconnected nodes cannot maintain their weight. Details about selective activation are presented in Section 4.4.2.

In the following, we discuss various forms of IncGraph annotations.

### 4.2.6 Self References

Self-reference annotations are added to IncGraph( $R$ ) for each node that represents a table. This annotation mitigates differences that can occur if an ontology uses multiple classes to model the content of one table of the relational schema. When these different entity types are in a 1 : 1 relation in the database, then it is likely that there are also object properties in the ontology that relate them. Note, that this may even be the case for properties that are not restricted as being functional and inverse functional, as the restriction of the relationship as 1 : 1 could be enforced only in the technical view of

the database (e.g., because this restriction is for the database application and applying it simplifies the schema). The same property might be non-functional from a semantic point of view in the general case. In either case, such relations are implicit in a relational database, as information about related entities are already stored in the same tuples of the same table. Therefore, no corresponding graph structure could be found between basic IncGraphs for such relations.

(A) Ontology  $O$ 

<b>Paper</b>
<u>ID</u>
title
abstract_content

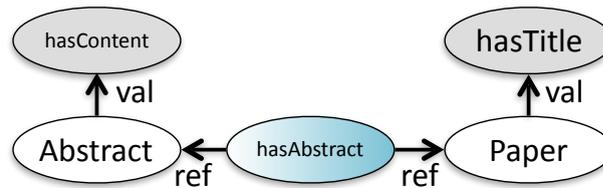
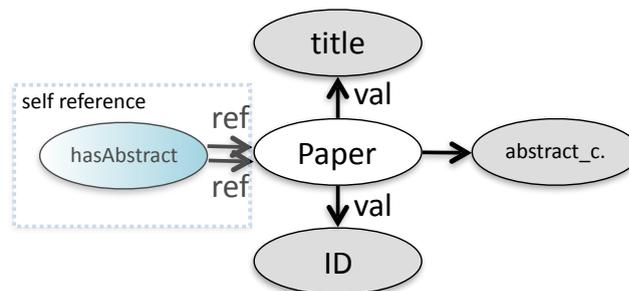
(B) Relational schema  $R$ (c)  $\text{IncGraph}(O)$ (D)  $\text{IncGraph}(R)$ 

FIGURE 4.7: IncGraph construction with papers and abstracts

Consider the ontology and relational schema shown in Figure 4.7, which follows our running Example 4.1 but adds *Abstracts* to *Papers* and leaves out the concept of *Authors*

for conciseness. Each paper has exactly one abstract. But whereas the abstract content in  $\text{IncGraph}(O)$  is a property of *Abstract* (and thus structurally separated from *Paper* by object node *hasAbstract*), it is a direct property of node *Paper* in  $\text{IncGraph}(R)$ . Thus, no structural correspondence could be found between the two. Adding a self-referencing node and edges, however, introduces a path in  $\text{IncGraph}(R)$  that can align with  $\text{IncGraph}(O)$  (shown in left part of Figure 4.7d). Node *Paper* in  $\text{IncGraph}(R)$  then aligns to both *Paper* and *Abstract* in  $\text{IncGraph}(O)$  and *hasAbstract* can be followed in a similar structural way on both sides. Note, that in Figure 4.7d we have labeled the self-referencing node *hasAbstract* to highlight the similarities between both  $\text{IncGraph}$ s. In reality, relevant similarities are merely structural and self-referencing nodes have empty labels.

This example shows that self-referencing annotations can result in a better structural similarity of  $\text{IncGraph}(O)$  and  $\text{IncGraph}(R)$ .

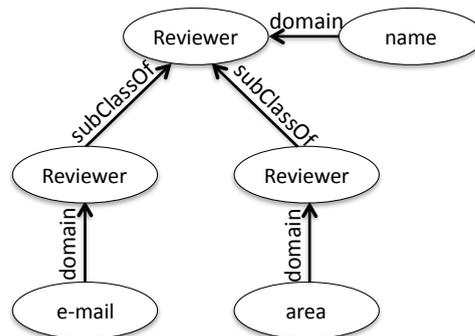
We repeatedly encountered such situations in practice. Still, self-references are added in an inactive state.

#### 4.2.7 Mapping Corresponding RDB2RDF Patterns

Although  $\text{IncGraph}$  narrows the gap in representation between relational schemata and ontologies as discussed above, differences still remain. Those are, by and large, again due to fundamentally different design patterns used on both sides. Where those patterns become more elaborate than the ones introduced above and encode complex semantics, simple and generic graph construction rules will no longer suffice to successfully map between them. In such cases, a correct mapping will need to be supported by knowledge about more complex mapping patterns.

Due to the different data modeling approaches, the same information is structured differently in relational databases and ontologies, and direct correspondences between modeling constructs are often difficult to establish. For instance, relational database schemata do not model hierarchies of concepts and properties explicitly, many-to-many relations are expressed using intermediate tables, etc. There exist at least three different ways in which an ontological *subClassOf* relation can be emulated in a relational model (Figure 4.8). In the database community, research has identified common design patterns for modeling interrelated data [28, Chap. 4.5f], and the Semantic Web research in turn noted some common ways for matching these database schema structures with the semantically equivalent ontology constructs [98, 135].

Such common correspondence patterns provide valuable background knowledge that helps to generate mappings between ontologies and relational databases. A specific subset of initial mappings can be reinforced if they appear to satisfy a convincing pattern.



(A) Ontology

**Relational Schema (Option 1)**

pid	name	e-mail	area	type
1	Lennon	a@b	-	author
2	McCartney	c@d	-	author
3	Harrison	-	Onto	reviewer
4	Starr	-	DB	reviewer

*Person***Relational Schema (Option 2)**

aid	name	e-mail	rid	name	area
1	Lennon	a@b	1	Harrison	Onto
2	McCartney	c@d	2	Starr	DB

*Author**Reviewer***Relational Schema (Option 3)**

pid	name
1	Lennon
2	McCartney
3	Harrison
4	Starr

*Person*

pid	e-mail
1	a@b
2	c@d

*Author*

pid	area
3	Onto
4	DB

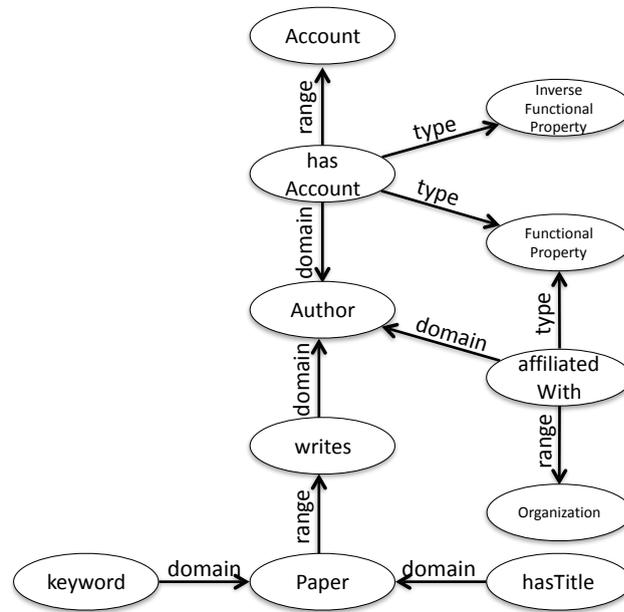
*Reviewer*

(B) Relational schema options

FIGURE 4.8: Correspondence patterns: class-subclass hierarchy

Based on common structures listed in the literature, we have identified the following patterns that are relevant, could directly be used, and are not already covered by our basic IncGraph or by basic unifying annotations:

- Subclass relations between classes:
  - *One common table for all.* An example is shown as Option 1 in Fig. 4.8. Here one table stores information about all people: both authors and reviewers.



(A) Ontology

aid	name	affiliated
1	Lennon	1
2	McCartney	1

**Author**

aid	login
1	lennon

**Account**

aid	pid
1	1
1	2
2	1

**Paper\_Author**

oid	name
1	Y.Submarine
2	fluidOps

**Organization**

kid	pid	title
1	1	Ontos
2	1	R2RML

**Paper\_Keywords**

pid	title
1	OBDA for LCD
2	R2RML for SF

**Paper**

(B) Relational schema options

FIGURE 4.9: Correspondence patterns: properties

The *type* column stores the type of the record, and the type-specific fields, such as *e-mail* and *area* receive NULL values if they are not relevant for the record.

- *Separate unrelated tables.* Option 2 in Fig. 4.8 provides an illustration: authors and reviewers are stored in separate tables containing only fields relevant for the specific type. Common fields such as *name* are contained in both tables.
- *Separate tables linked via a 1 : 1 foreign key relation.* See Option 3 in Fig. 4.8: common fields are defined in the Person table, which has 1 : 1 foreign key relations to both category-specific ones.

- Object property links:
  - *1 : 1 relation*: two tables are connected via their unique keys, similarly to the relation between the tables Author and LoginAccount in Fig. 4.9.
  - *1 : n relation*: two tables are connected via a foreign key, which is unique in one of the tables, as is the Organization table in Fig. 4.9. The unique key *oid* of the Organization table is referenced in the Author table.
  - *n : m relation*: two tables are connected via an intermediate table containing two foreign key columns. The PaperAuthor table in Fig. 4.9 is used as such an intermediary between the Author and Paper tables.
  - *object join path*: where a sequence of connected tables shows a pattern auf a plausible join path, this may correspond to a single object property, often (but not necessarily) in a *n : m* relation.
  - *transitive relation*: independent of cardinalities, if a property is known to be transitive it might likely correspond to a recursive pattern in the database (not shown in the figures), i.e., where a non-key attribute references a key of the same table.
  
- Datatype property links:
  - *1 : 1 relation*: a column in the table directly contains the value of a datatype functional property (e.g., *hasTitle*).
  - *1 : n relation*: a separate table is linked via a foreign key to the main table and contains an additional column for the data values. The example is the PaperKeywords table in Fig. 4.9.
  - *data join path*: where a sequence of connected tables shows a pattern auf a plausible join path, some data might be reachable through this path, only. It thus could correspond to a (likely non-functional) datatype property.

Note, that this list is not exhaustive but based on previous analyses on most relevant patterns ([28, 98, 135]). It is also limited to those cases that are not already generically covered by the basic IncGraph model or other unifying annotations discussed earlier. Also note, that these patterns do not deterministically imply the specific semantics that our list of correspondences suggests. That is, they are merely heuristics, which can be leveraged to identify *likely* correspondences, which could otherwise not be matched with IncGraph. Our approach to implementing support for such patterns is therefore designed to be flexible, self-adjusting and extensible.

### 4.2.8 Ontology Meta Knowledge Annotations

As discussed in previous Section 4.2.7, even complex patterns in a relational schema often correspond to a single axiom on the ontology side. In fact, although there are many more complex design patterns in ontologies as well, all of the relevant relational patterns listed above can be seen as correspondences to individual axioms.

This is why, when constructing IncGraphs from ontologies, we mostly use a selection of these axioms to construct pattern type nodes in  $\text{IncGraph}(O)$ . Note, that we call the corresponding IncGraph nodes *pattern nodes* (axiom occurrences) and *pattern type nodes* (different axiom types) even on the ontology side. This way, we end up with identical types of nodes in both  $\text{IncGraph}(R)$  and  $\text{IncGraph}(O)$  to support matching.

In addition to classes, object properties and datatype properties, which are already modeled by basic IncGraphs, we annotate the following types of OWL 2 axioms, whenever available:

- As potential correspondences for hierarchy patterns:
  - SubClassOf (directly indicating object hierarchies)
  - SubObjectPropertyOf (indirectly indicating object hierarchies)
  - UnionOf, DisjointUnionOf (indicating potential anonymous super types)
- As correspondences for cardinality patterns:
  - FunctionalObjectProperty, FunctionalDataProperty (indicating  $n : 1$  relations)
  - InverseFunctionalObjectProperty (indicating  $1 : n$  relations)
  - ObjectMinCardinality, DataMinCardinality, ObjectMaxCardinality, DataMaxCardinality, ObjectExactCardinality, DataExactCardinality (indicating cardinalities depending on exact values and combination)
  - HasKey (an obvious candidate in theory, which we have never observed in actual ontology that we were testing)
  - Properties in the absence of any of the above (lack of restriction may indicate  $n : m$  in some cases)
- As correspondence candidates for recursion patterns:
  - TransitiveObjectProperty
  - SymmetricObjectProperty

Again, the above list is not exhausting and we keep the system open to additional axioms by making this list configurable in the implementation. Note, that the grouping above into classes of potential correspondences is indicative in nature but does not impose any limitations of potential pattern correspondences. In principle, any pattern type node in  $\text{IncGraph}(O)$  can align with any pattern type node in  $\text{IncGraph}(R)$  if repeated co-occurrences suggest such an alignment.

As a precondition for leveraging knowledge from the ontology during matching, relevant axioms need to be captured in  $\text{IncGraph}(O)$ .

Let  $u \in U$  be a unary ontology axiom that appears in  $O$ , and let  $v_a \in V$  be a any node constructed from  $a \in O$ . Then:

#### Unary Axiom Nodes:

$$\begin{aligned} u(a) \rightarrow v_y \in V \wedge v_y.type = "Y" \wedge e_y = (v_a, v_y) \in E \wedge (e_y, "pi") \in lbl_E \\ \wedge v_x \in V \wedge v_x.type = "X" \wedge (v_x, name(u)) \in lbl_V \\ \wedge e_x = (v_y, v_x) \in E \wedge (e_y, "pt") \in lbl_E \end{aligned}$$

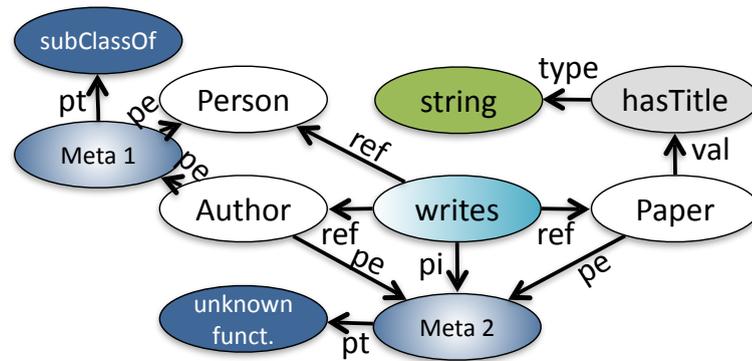
Technically, in addition to inner pattern edges ( $pi$ ), pattern endpoint edges ( $pe$ ) are also constructed to connect with elements that are immediate neighbors of the element on which the unary axiom applies. For better readability we do not reflect this addition in the above formula.

Let  $b \in B$  be a binary ontology axiom that appears in  $O$ , and let  $v_{a_1}, v_{a_2} \in V$  be nodes constructed from  $a_1, a_2 \in O$ . Then:

#### Binary Axiom Nodes:

$$\begin{aligned} b(a_1, a_2) \rightarrow v_y \in V \wedge v_y.type = "Y" \\ \wedge e_y = (v_{a_1}, v_y) \in E \wedge (e_y, "pe") \in lbl_E \wedge e_y = (v_{a_2}, v_y) \in E \wedge (e_y, "pe") \in lbl_E \\ \wedge v_x \in V \wedge v_x.type = "X" \wedge (v_x, name(b)) \in lbl_V \\ \wedge e_x = (v_y, v_x) \in E \wedge (e_y, "pt") \in lbl_E \end{aligned}$$

Figure 4.10 shows a full  $\text{IncGraph}(O)$  from running Example 4.1 with annotations of ontology meta knowledge in pattern style. Besides all of the nodes that are already present in the basic  $\text{IncGraph}$ , nodes from two additional axioms have been added. At the bottom, property *writes* is marked as an unrestricted property, which may indicate

FIGURE 4.10: Full regular  $\text{IncGraph}(O)$ 

that it is used as an  $n : m$  property. A *pattern* node (*Meta 2*) is therefore included in the graph. Also, this node is connected to a *pattern type* node (*unknown funct.*). This is a unary axiom, and a meta node (vertex type *pattern*) has been directly connected to the node representing the property (*pi*-edge). Additional *pe*-edges are supporting the scope of the pattern. Those edges help to smoothen alignment with corresponding patterns that may be less atomic in structure. On the left side of the figure, *Author* is modeled as a subclass of *Person*, which is represented by a binary axiom pattern connecting the two.

Note, that this two-stage modeling of *pattern type* nodes with intermediate *pattern* nodes is necessary if several occurrences of the same axiom appear in the ontology. In this case, all *pattern* nodes of the same axiom connect to the same *pattern type* node. This leads to unique axiom *pattern types* to align well with corresponding *pattern type* nodes on the relational side exactly if several of their *pattern* node instances co-occur in similar context on both sides. Also note, that meta nodes technically have empty labels.

#### 4.2.9 Reasoning on Ontology Annotations

Reasoning can add to the value of ontology  $\text{IncGraphs}$ .

It is important to understand, though, what reasoning in the context of  $\text{IncGraph}$  construction means and which roles inferred axioms or facts may play. Any axioms or facts that will reflect into an  $\text{IncGraph}$  will be represented as merely structural graph features. This is the same as for any other modeling elements of the input ontology. As structural graph features, they serve one single purpose: to act as match *candidates* during graph matching. Adding missing features is therefore helpful if and only if they represent correct matches for corresponding features in the other  $\text{IncGraph}$ . If such graph features, however, they act as noise and are potentially harmful for matching quality, even if the axioms or facts that they represent are correct. Thus, we need to be careful about

activating reasoning before the graph matching phase. It is also important to note that the decision for or against reasoning at this stage does not preclude any later reasoning on the resulting data.

When building a graph, reasoning can be applied at two different points in the process: immediately before IncGraph construction, i.e., on an original input ontology, or during IncGraph construction, i.e., using custom rules that operate directly on the IncGraph model. Both of these options cause different effects.

Firstly, a reasoner could infer and materialize relevant T-Box axioms prior to IncGraph generation. While obvious at first glance, this may in principle be an important step to guarantee that all relevant nodes can be created during graph construction. Unfortunately, in practice the approach also has several downsides: a few correct candidate correspondences might be added, but many more unnecessary and potentially misleading candidates could be added as well. In a number of initial tests that we conducted with standard OWL QL reasoning enabled in this way, we did observe a generally detrimental effect on mapping quality due to noise. One case where noise is a particularly troubling problem is with the interpretation of concrete domains and ranges, which are crucial for structural matching. On top of that, many ontologies are not designed as cleanly as one might hope, adding the risk of inferring incorrect axioms. For these reasons, we do not recommend to apply this step, unless the ontology in question is known to be clean and also uses precise domain and range definitions.

Secondly, we derive implicit information using custom reasoning rules and directly encode consequences into  $\text{IncGraph}(O)$ . Besides limiting the reasoning scope to select rules, the main difference here is how we treat reasoning results as opposed to explicitly stated axioms. More specifically, we derive implicit subclass axioms as well as equivalences for both classes and properties. However, for all transitive calculations (e.g., subclass of subclass), we weigh down edges by slashing their weight in half with each additional step down the path. This follows the intuition that axioms further away from what has been explicitly declared are also less likely to be explicitly modeled in a corresponding relational database. We additionally derive disjoint unions of classes in some cases. Similarly, for properties, we derive axioms about their functionality as well as information about their domains and ranges, although not exhaustingly. We eventually also remove symmetric pairs of subclass axioms and superclass axioms (i.e., implicit equivalences) to avoid clutter in the resulting graph. By default, IncMap only employs this form of built-in reasoning.

This way, we limit reasoning on IncGraph to a very small number of cases where we have positively observed before that they are likely to be needed during structural graph matching. In addition, by reducing the weight of edges in some cases, we reduce the

noise effect of graph features that may be very helpful at times, but in relatively fewer cases.

In the following, we discuss the non-standard reasoning rules applied in this phase.

#### 4.2.9.1 Sub Classes and Super Classes

Domains and ranges are often modeled in a database at a granularity other than the one expressed in a corresponding ontology. In one case, this may be one or more specific subclass(es) of the actual domain or range, if the database is designed to accept only information about those specific subclasses. In another case, however, this can even be a *super* class of the domain or range, following the reasoning that *some* of the individuals of the super class can have values of that property. Although somewhat less frequent, databases occasionally happen to model properties in such an overly generic way.

In order to solve this, we add edges to encode all alternative connections. To express the notion of more unexpected cases, we assign weight factors to additional edges:

##### Sub Classes:

$$\begin{aligned} &v \in V \wedge v_d.type \in \{ "A", "R" \} \wedge e = (v, v') \in E \wedge v'.type = "T" \\ &\wedge (e, l_e) \in lbl_E \wedge (e, w_e) \in W_E : \forall s \in C : subClass(class(v), s) \\ &\rightarrow e' = (v, node(s)) \in E \wedge (e', l_e) \in lbl_E \wedge (e', w_e/2) \in lbl_E \end{aligned}$$

##### Super Classes:

$$\begin{aligned} &v \in V \wedge v_d.type \in \{ "A", "R" \} \wedge e = (v, v') \in E \wedge v'.type = "T" \\ &\wedge (e, l_e) \in lbl_E \wedge (e, w_e) \in W_E : \forall s \in C : superClass(class(v), s) \\ &\rightarrow e' = (v, s) \in E \wedge (e', l_e) \in lbl_E \wedge (e', w_e/4) \in W_E \end{aligned}$$

Technically, to calculate super classes, we look at *subClassOf* axioms and apply edge construction in inverse direction.

#### 4.2.9.2 Equivalence and Pseudo Equivalence

We infer a number of equivalences using standard reasoning rules immediately before graph construction.

While real equivalences can also be made explicit by a reasoner in pre-processing, another, subtler notion of equivalence may apply on IncGraph for axioms that are not even equivalent in the ontology. We call this phenomenon *pseudo-equivalences*. It is caused by relations and referential constraints having no semantic direction and because, in IncGraph, we only model aspects that can find correspondences. As a consequence, ontology properties also lose their direction in IncGraph w.r.t. matching. Therefore, inverse properties become effectively equivalent in IncGraph. However, they are not equivalent in the underlying ontology and thus are normally modeled separately during graph construction. In this case they compete for matches and distract structural alignment.

We solve this issue by unifying pseudo-equivalent property axioms into a single node during IncGraph construction, but maintain both labels for alternative lexical matching.

Note, that no formal node construction rule for this modification can be given here, as this is a replacement operation modifying existing nodes in the graph, not an addition of extra nodes.

#### 4.2.9.3 Deprecated Classes and Properties

OWL supports deprecation in form of a dedicated annotation property. In an ideal world, deprecated features in an ontology should no longer be used for querying and might have been replaced by other elements. Under this assumptions, deprecated elements should not become part of IncGraph to avoid introducing unwanted matches. More realistically, however, this conclusion can only *sometimes* be drawn for deprecated ontology elements, not *always*.

We therefore attempt a compromise by post-processing nodes constructed for deprecated ontology elements and by reducing the weight of all their edges by 50%.

#### 4.2.10 Relational Pattern Annotations

For relational schemata, IncMap exploits design patterns with heuristic rules that enrich the IncGraphs with *pattern* nodes. Such pattern nodes represent a specific design pattern (e.g., *Class-SubClass*) and are connected by edges to the relevant content nodes. In contrast to axioms in the ontology, patterns in the database are merely structural features, which may or may not represent the assumed semantics. In addition, some patterns are also ambiguous (e.g., 1 : 1 relations vs. the third example option for modeling hierarchies in relational databases). Also, a number of patterns cannot even be identified with certainty, but heuristics apply and lead to varying confidence scores. Thus,

on the database side, we employ weighted edges to connect pattern nodes, representing their detected confidence score. The role of the pattern nodes is to reinforce the connection between semantically similar aspects on both sides, i.e., between  $\text{IncGraph}(R)$  and  $\text{IncGraph}(O)$ .

Formally, for each supported pattern  $P$  on relational schema  $R$ , there is a pattern qualifier heuristic  $H_P$  that assigns each subset of schema elements (i.e., each  $S \in \mathcal{P}(R)$ ) a score, denoting the confidence that they might form the specified pattern.

### Pattern Nodes:

$$\begin{aligned}
 H_P(S) > 0.0 &\rightarrow \forall el \in S : v_y \in V \wedge v_y.type = "Y" \\
 \wedge e_y &= (v_{el}, v_y) \in E \wedge (e_y, "pi") \in lbl_E \\
 \wedge v_x &\in V \wedge v_x.type = "X" \wedge (v_x, name(P)) \in lbl_V \\
 \wedge e_x &= (v_y, v_x) \in E \wedge (e_x, "pt") \in lbl_E \wedge (e_x, H_P(S)) \in W_E
 \end{aligned}$$

Technically, instead of inner pattern edges ( $pi$ ), pattern endpoint edges ( $pe$ ) are constructed to connect with elements that have at most one connection with other elements belonging to the pattern. For brevity, we do not reflect this distinction in the above formula.

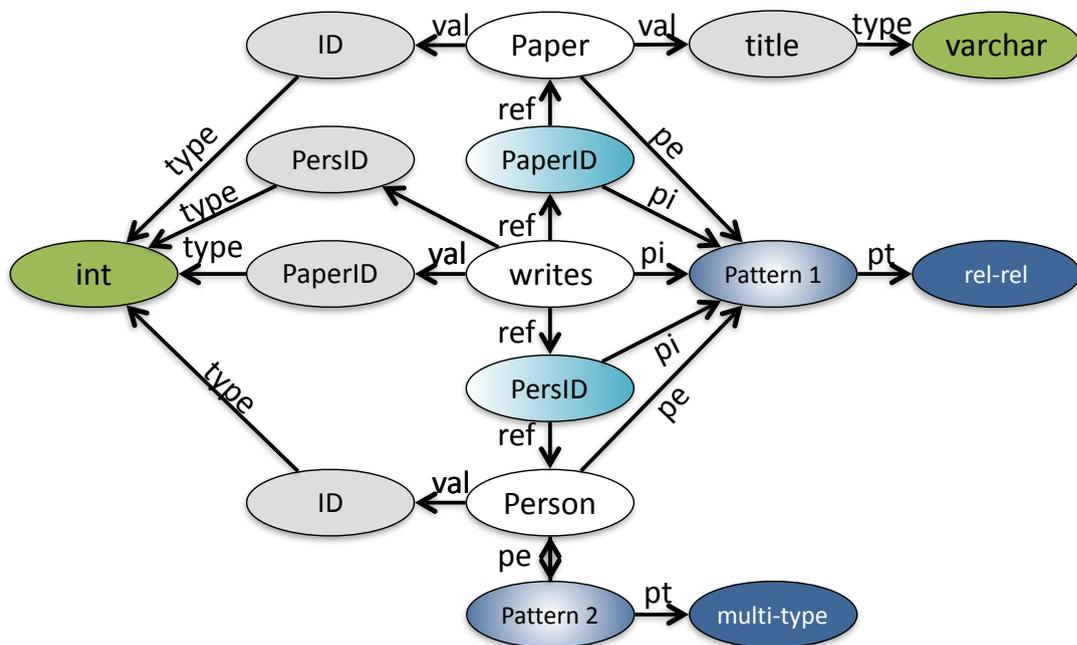


FIGURE 4.11: Full regular  $\text{IncGraph}(R)$

Figure 4.11 shows a full  $\text{IncGraph}(R)$  from running Example 4.1 with annotations of relational patterns. Two pattern types have been added, *rel-rel* (indicating a possible

relationship relation) and *multi-type* (indicating a table that likely contains tuples describing more than one entity type). For each pattern type, exactly one instance has been observed.

If you relate this graph with the corresponding  $\text{IncGraph}(O)$  from Figure 4.10, it is easy to see how pattern type *rel-rel* could align with *unknown funct.*, while *multi-type* may align with *subClassOf*.

### 4.2.11 Shortcut Edges

Even full  $\text{IncGraphs}$  can be further optimized with additional annotations.

While patterns can help to support alignments of related node groups, constructing complex correspondences from two graphs still requires fully corresponding sub graphs at both ends. This is often not possible, because patterns have been identified but look structurally different.

For instance, the path from *Person* to *Paper* in Figure 4.11 is still longer than the path from *Person* (or from *Author*) to *Paper* in Figure 4.10. A correspondence for property *writes*, which connects the two, can therefore not be identified immediately although it is supported by an aligned pattern.

In order to mitigate these differences and support arbitrary correspondences across patterns we add shortcut edges to  $\text{IncGraph}(R)$  that represent potential join paths.

Figure 4.12 illustrates how shortcuts could solve this issue. The additional shortcut path has the same structural properties as the property path in the corresponding  $\text{IncGraph}(O)$  (Figure 4.10).

Adding shortcut edges everywhere where they potentially could make sense is dangerous, though. Their introduction is always only one possibility among others, and adding them carelessly adds a lot of noise to the matching graphs. We therefore add shortcut edges in an inactive state and selectively activate them later on, only when they have strong support by their surrounding patterns.

### 4.2.12 Annotations from User Queries

In ontologies, the range of an object property is often not precisely modeled. On the one hand, this leads to greater flexibility in the application of these properties, which is a particularly desirable feature to enable the re-use of ontologies. On the other hand, it



### 4.2.13 Implementation Notes on IncGraph Construction

IncMap requires information about input schemata to be added through its dedicated API, i.e., IncGraph construction is passive.

Graph construction is implemented in IncMap using general-purpose graph structures for directed labeled graphs built on indexed Java Collections. Labels can be stacked and implement a minimal common interface only to be extensible to a degree where they could contain, among other things, provenance information, translation methods or even partial sub graphs. On this basis, IncGraph vertices and edges are being used as building blocks also for all other graph structures during matching. We have taken this decision to optimize overall memory consumption and to maintain a high degree of flexibility in implementing different algorithms on top of the graph structures. It comes at the price of a relatively high initialization overhead, though.

We augment IncGraph nodes with rich provenance information. For IncGraph ( $O$ ), this includes the target axioms, but also additional context in some cases, e.g., and interpretation of the most specific domain and range for properties. For IncGraph ( $R$ ), provenance essentially describes a full Select-Project-Join (SPJ) access path to the data described. Although this access path is stored in a dedicated data structure designed for IncMap, it essentially expresses relational SPJ queries, although restricted to equi-joins and with limited selection capabilities. Provenance information on the source schemata can be automatically translated into SQL queries or partial SQL queries.

Edge weights are technically assigned based on configurable weight factors. The fixed weight factors that we been reporting in this section are default choices, which we have observed to work reasonably well in many example situations.

## 4.3 Matching Background: Similarity Flooding

In this section we briefly summarize the original Similarity Flooding algorithm for schema matching as described by Melnik et al. [45, 54].

The starting point of the Similarity Flooding algorithm are two directed labeled graphs, which represent the schema elements of a source schema and target schema to be matched. Each graph is defined as a tuple  $G = (V, E)$  of vertices  $V$  and edges  $E$ , where each edge  $e \in E$  is represented as a triple  $(s, p, o)$  with  $s \in V$  being the source vertex,  $o \in V$  the target vertex and  $p$  the label of the edge.

The procedure to construct an input graph from a given schema is not defined by [45]. Thus, the Similarity Flooding algorithm is open for any graph construction process

and for any graph structure, as long as it adheres to the model described above. It is therefore up to the implementation, which vertices, edges, and labels are created for a given schema.<sup>3</sup>

To explain the further process, we use the same example as the one presented in [45]. Figure 4.13 depicts this example. The two input graphs shown on the left hand side (Figures 4.13a and 4.13b) represent the structure of a given source and target schema, respectively.

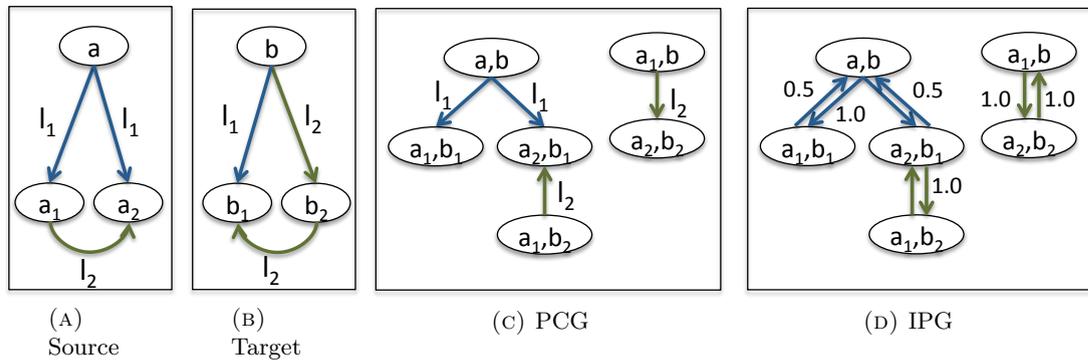


FIGURE 4.13: Similarity Flooding

As a next step in the Similarity Flooding algorithm, a so called *pairwise connectivity graph* is built from the constructed source graph  $G_S$  and target graph  $G_T$ . The pairwise connectivity graph (PCG) is defined as follows: nodes are elements from  $V(G_S) \times V(G_T)$ , representing potential match candidates (called match pairs in a  $PCG(S, T)$ ). For example, the node  $(a, b)$  in the example PCG (Figure 4.13c) represents a potential match candidate of node  $a \in G_S$  and  $b \in G_T$  of the input graphs. Moreover, an edge of the form  $((x, y), p, (x', y'))$  between any two nodes  $(x, y)$  and  $(x', y')$  of a PCG is added if  $(x, p, x') \in E(G_S) \wedge (y, p, y') \in E(G_T)$ . The intuition behind the edges is to model structural commonalities in both graphs  $G_S$  and  $G_T$  in the  $PCG(G_S, G_T)$ . In the example PCG of Figure 4.13, the match pair  $(a, b)$  has an edge to the match pair  $(a_1, b_1)$  with the label  $l_1$  since both nodes  $a$  and  $b$  have an outgoing edge  $l_1$  in the given source (target) graph to the node  $a_1$  ( $b_1$ ).

Based on the  $PCG(G_S, G_T)$  a so called *induced propagation graph* (IPG) is derived. The structure of  $IPG(G_S, G_T)$  is similar to that of  $PCG(G_S, G_T)$ . The only difference is that an additional edge in the opposite direction is added for each edge of  $PCG(G_S, G_T)$ . Moreover, for each edge of  $IPG(G_S, G_T)$  a weight is attached (called propagation coefficient). The weight of an outgoing edge  $w(e)$  with label  $l$  (of a given node) is calculated by  $w(e) = 1/out_l$  where  $out_l$  is the number of outgoing edges with the same label  $l$ . In

<sup>3</sup>[54] describes the authors' assumptions for relational schema matching in some detail. They are not part of a defined algorithm but rather presented as an example, though.

the example IPG in Figure 4.13d the weights for the outgoing edges with label  $l_1$  of node  $(a, b)$  are both set to 0.5, since the node has two outgoing edges with that label and has to split its weight among them.

The final step of the Similarity Flooding algorithm is a fixpoint computation to propagate initial similarities through the graph by using the structural dependencies. Therefore each matching pair  $(x, y)$  of an IPG is initialized with its initial similarity  $\sigma^0(x, y)$  by using a lexical similarity matcher. Then, the initial similarities are adopted by a step-wise computation: in every iteration  $i$ , with  $i \geq 1$ , a new value of  $\sigma^i(x, y)$  is computed for any match pair  $(x, y)$  by incrementing its similarity value of the previous iteration by the  $\sigma$ -values of its neighbor pairs in the IPG multiplied by the propagation coefficients on the edges going from the neighbor pairs to node  $(x, y)$ .<sup>4</sup>

The idea is that a matching pair  $(x, y)$  of an IPG profits from the structural similarity represented in the IPG by the edges. For example, if we assume that  $\sigma^0(x, y) = 1$  for any match pair in the IPG of Figure 4.13, then  $\sigma^1(a_2, b_1) = \sigma^0(a_2, b_1) + 0.5 * \sigma^0(a, b) + 1.0 * \sigma^0(a_1, b_2) = 1 + 0,5 * 1 + 1 * 1 = 2.5$ .

At the end of each iteration, the  $\sigma^i(x, y)$  value for each node  $(x, y)$  is normalized by the maximal  $\sigma$ -value of this iteration (i.e., all  $\sigma$ -values are  $\leq 1$  and  $\geq 0$ ). The fixpoint computation terminates if either the residual vector  $\delta(\sigma^n, \sigma^{n-1})$  becomes less than a given  $\epsilon$  or a maximum number of iterations  $n$  has been performed.

## 4.4 IncMap Matching

### 4.4.1 Basic Matching Process

Basic, non-incremental correspondences and mapping suggestions result from lexical matches of the input IncGraphs that are then being re-ranked based on the structure of their combined matching graph. This process mostly follows the Similarity Flooding [45] algorithm.

Matching between a source and target IncGraph starts with calculating the Cartesian products between nodes in the source and target, for each respective node type or color. Essentially, we use a technique from Similarity Flooding with the two input graphs (IncGraph( $R$ ) and IncGraph( $O$ )), similar to the construction of PCGs as described in Section 4.3.

<sup>4</sup>Some alternative propagation models are also discussed in [45, 54].

An initial match operator then evaluates each pair of nodes, i.e., each potential match. Initial matching calculates a lexical similarity score between the nodes, using one of several interchangeable matchers. The primary source of information for lexical matching are node labels, which normally result from schema element identifiers.<sup>5</sup>

TABLE 4.1: IncMap lexical matchers (default in bold print)

Matcher	Short Description	Label	Document
LS	Classic Levenshtein similarity (inverse edit distance)	yes	no
XLS	Levenshtein variant (prefers distinctive scores)	yes	no
ID	Identity matcher (string equality)	yes	no
AEQ	All-equal at 1.0 (experimental baseline)	no	no
WB	Document word bag Jaccard similarity	no	yes
MS	Multi-source LS and WB	LS	WB
TWB	String tokenizer, WB on tokens	yes	no
<b>STB</b>	Stemmed, stop-word filtered, tokenized WB	yes	no

Table 4.1 shows a list of all our supported lexical matching operators. Each matcher is listed with its identifier, a short description, and its basic modes of operation: matchers could either work on labels, on the overall document of words drawn from additional annotations, or on a combination of both. The table lists match operators in the historic order of implementation: in IncMap’s first version (as described in the original IncMap paper [52]), the operator list was implemented from top down to *AEQ*. Those were mostly simple operators. For instance, Levenshtein similarity (*LS*), the inverse of the Levenshtein distance [136], is a traditional string similarity metric but not highly performing. The all-equal matcher (*AEQ*) assigns a uniform default score to all nodes without even looking at their labels or annotated documents. It serves as a baseline measure for debugging and to demonstrate the viability of structural matching alone, i.e., without effective support of any initial lexical matchers.

Operators can be selected by configuration. Our preferred and default initial match operator is word-bag Jaccard similarity on stemmed, stop-word filtered tokenizations (*STB*). It is both highly performing and has proven to be highly effective in most of our experiments. This operator uses label information, only. For nodes with no lexical labels, such as pattern nodes, the operator assigns an initial similarity of 0.5.

Once lexical matching is complete, paired nodes get reassembled into a new graph. Graph reassembly is based on common edges that both paired nodes used to share in their respective IncGraphs. For instance, if ontology node *A* has a *ref* edge to ontology node *B* in IncGraph(*O*), and also relational schema node *X* has a *ref* edge to schema

<sup>5</sup>On the ontology side, *rdf:labels* are preferred, where available. We technically also support to match on an annotated *node document*, which is a concatenation of annotation properties *rdfs:comment*, *rdfs:seeAlso*, and *rdfs:isDefinedBy* for the ontology and PostgreSQL *COMMENT* text in case of relational schemata. Textual annotations for schemata other than PostgreSQL are not currently supported.

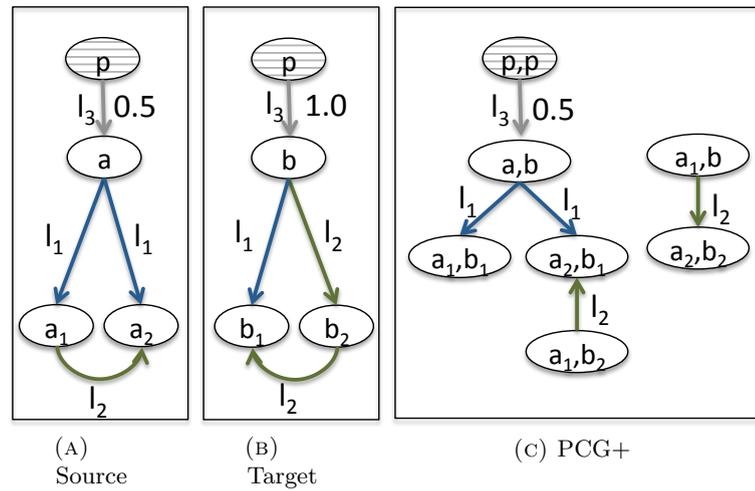


FIGURE 4.14: Construction of PCG+ (Simplified)

node  $Y$  in  $\text{IncGraph}(R)$ , then we will add a *ref* edge between paired nodes  $(A, X)$  and  $(B, Y)$ .

Pairing of nodes results in an Extended Pairwise Connectivity Graph (PCG+). Figure 4.14 shows a generic and simplified example of two input graphs and the resulting PCG+ (Figure 4.14c). The PCG+ has a node for every pair in any of the cross products of nodes from the input graphs, where nodes of some type (or color) in one graph will be paired only with nodes of the same type in the other. Therefore, in the figure, all combinations of  $a, a_1, a_2, b, b_1$  and  $b_2$  are in the graph, but only one node pairing  $p_1$  with  $p_2$ . Edges are added to the PCG+ wherever both constituent nodes of a pair had a shared type of edge in the same direction. For instance,  $a$  and  $b$  have both an outgoing edge labeled  $l_1$ , which leads to  $a_1$  and  $b_1$ , respectively. Therefore, the pair  $(a, b)$  in the PCG+ also has an outgoing edge  $l_1$  leading to the pair  $(a_1, b_1)$ . In addition, if either edge was weighted, their weights get multiplied for the PCG+. Unweighted edges are assumed to carry a weight of 1.0. If either edge is inactive and therefore optional, the common edge becomes inactive.

Our graph is based on the original PCG from [45] but extended in several ways, primarily to accommodate weighted edges and optional edges, which are not supported in the original PCG.

Weighted edges are our means to connect  $\text{IncGraphs}$  in uncertain ways and factor that uncertainty into the graph structure. Other than adjusting the scores of nodes that those edges connect, adding an edge weight allows us to gain more control over the graph flow in the following fixpoint computation. Additionally, while node scores interact with their neighborhood and can change quickly and unexpectedly during the fixpoint computation, edge weights remain unchanged.

Another difference is the handling of inactive *ref* edges in the input IncGraphs. For inactive *ref* edges, which are not known to the original Similarity Flooding, we apply the following rule when building the PCG+: if an edge in the PCG+ refers to at least one inactive *ref* edge in one of the IncGraph models, it also becomes inactive in the PCG+.

PCG+ generation is to prepare for a following phase, where a fixpoint computation based on the Similarity Flooding algorithm [45] refines the initial scores based on structural similarities.

The PCG+ is then transformed into the final input for Similarity Flooding by adding inverse edges to avoid black holes in the fixpoint computation and by (re-)assigning weights to all edges to balance the score distribution. At this point, user feedback or partial mappings can be incorporated by adjusting the initial scores of matches to 0.0 or 1.0 and by optionally flagging them as stable (i.e., their score will remain unaffected throughout the fixpoint computation and therefore exerts a more constant influence on their structural neighborhood).

Finally, the fixpoint computation will repeatedly distribute scores of matches to neighboring nodes. Distribution of scores depends on edge weights, thus refining the score of correspondences structurally.

These last steps can be repeated several times for several iterations of user feedback.

One more difference between IncMap and original Similarity Flooding relates to propagation coefficients. In IncMap they are modular and can be changed by configuration. In particular, a new weighting formula supported by IncMap considers the similarity scores on both ends of an edge in the IPG. The intuition behind this is that a higher score indicates better chances of the match being correct. Thus, an edge between two matches with relatively high scores each is much more relevant for the structure than edges between one isolated well-scored match and several ones with extremely poor scores. For calculating the weight  $w(e)$  of a directed edge  $e = (n_1, n_2)$  from  $n_1$  to  $n_2$  in the IPG where  $l$  is the label of the edge, IncMap can use one the following alternatives:

- *Original Weight as in [45]:*  $w(e) = 1/out_l$  where  $out_l$  is the number of edges connected to node  $n_1$  with the same label  $l$
- *Similarity Product:*  $w(e) = score(n_1) * score(n_2)$  where  $score(n)$  is the score of the initial lexical matching of represented by node  $n \in IPG$ .
- *Normalized Similarity Product:*  $w(e) = (score(n_1) * score(n_2))/out_l$ .

Of these, the normalized similarity product is our default.

#### 4.4.2 Extended IncMap Matching: Edge Activation

The first extension of the basic version of our matching process is to activate inactive *ref* edges in the IPG before the fixpoint computation to be used for matching. In the following we describe the activation strategies of IncMap.

As described before in Section 4.2.5, inactive edges are annotations that might be structurally clarifying if applied carefully. However, since the number of possible edges is generally high, they are kept inactive until there is an actual reason to consider them.

To activate inactive edges, we introduce the concept of a *Node of Intense Interest* (NII). The intuition behind NIIs is to identify those match candidates in an IPG that most likely represent a correct correspondence between schema elements in *O* and *R*. IncMap supports two flavors of identifying NIIs: either a node in an IPG is an NII if it satisfies a certain similarity score threshold, or if it belongs to a set of *top-k* nodes with the highest matching scores in the IPG.

In order to activate an edge, we search for inactive edges (or edge paths, in the case of shortcuts) that connect two NIIs. Since both ends of the edge are NIIs, they indicate high quality matches. Intuitively, there are two potential policies as to when an inactive edge could be activated based on NIIs: either, if the shortcut edge connects two NIIs with each other (as described before), or, less strictly, if it connects an NII on one side with any arbitrary node on the other side. IncMap supports both of these modes, but recommends to use the strict approach.

Once inactive shortcut edges in an IPG are activated, IncMap starts or continues the Similarity Flooding fixpoint computation to leverage the new structural properties. During each step of the fixpoint computation, the match scores of nodes in the IPG might increase or decrease. As match scores change after each step of the fixpoint computation, IncMap also adapts the set of NIIs.

Consequently, for each potential match (i.e., a node in the IPG) that has changed its NII status after some step of the fixpoint computation, the set of activated edges needs to be adjusted based on the new set of NIIs. Alternatively, the set of NIIs can be kept stable after its first initialization, even while the matching scores in the IPG change and some initially selected NIIs do not satisfy NII criteria anymore. This procedure would significantly over-emphasize the influence of initial lexical matching, though. As soon as the fixpoint computation terminates, the IPG yields the final set of matches based on edge activation.

Edge activation leads to one more difference in IncMap over the original Similarity Flooding. Other than in the original approach, where propagation coefficients for the

IPG are ultimately determined during graph construction, our propagation coefficients can be calculated several times when the graph changes with the activation and deactivation of edges. This is necessary, as propagation coefficients depend on the number of edges of each type per node, which varies as edges get activated or deactivated.

### 4.4.3 Extended IncMap Matching: Iterative User Feedback

User feedback can be leveraged after any round of matching. Typically, this will happen in a semi-automatic mapping process, i.e., by offering the user *mapping suggestions*, which they accept or reject. Query-driven incremental mappings offer a particular opportunity to leverage user feedback interactively.

One of the reasons why we have chosen Similarity Flooding as a basis for IncMap is the fact that user feedback can be integrated by adopting the initial match scores in an IPG before the fixpoint computation starts or restarts.

Although the possibility of an incremental approach has been mentioned already in the Similarity Flooding paper [45], it has not previously been implemented and systematically evaluated. Also, while it is trivial to see *when* and where user feedback could be incorporated in the IPG, it is far less trivial to decide, *which* feedback should be employed and *how* exactly it should be integrated in the graph.

We focus on leveraging the most straight-forward and decisive kind of user feedback, i.e., the previous confirmation or rejection of suggested mappings. In addition, we accept partial existing mappings as implicit confirmation. Partial mappings could be the result of earlier iterations of an automatic approach or could have been hand-crafted.

We have devised three alternative methods how this kind of feedback could be added into the graph.

First, as a confirmed match corresponds to a certain score of 1.0, while a rejected match corresponds to a score of 0.0, we could simply re-run the fixpoint computation with adjusted initial scores of confirmed and/or rejected matches. We call this first method *Initializer*. However, there is a clear risk that the influence of such a simple initialization on the resulting mapping is too small as scores tend to change rapidly during the first steps of the fixpoint computation. For example, even a confirmed mapping would have no more influence on the resulting mapping than one of potentially many perfect string matches, some of which that may still be wrong.

To tackle this potential problem, our second method guarantees maximum influence of feedback throughout the fixpoint computation. Instead of just initializing a confirmed or

rejected match with their final score once, we could repeat the initialization at the end of each step of the fixpoint computation after normalization. This way, nodes with definite user feedback influence their neighborhood with their full score during each step of the computation. We therefore call this method *Self-Confidence Nodes*. As scores generally tend to decrease in most parts of the graph during the fixpoint computation and high scores become more significant for the ranking of matches in later fixpoint computation steps, this method implies the opposite risk of the Initializer method, namely, to over-influence parts of the graph. For example, one confirmed match in a partially incorrect graph neighborhood would almost certainly move all of its neighbors to the top of their respective suggestion lists.

Finally, with our third method, we attempt to balance the effects of the previous two methods. We therefore do not change a confirmed match directly but include an additional node in IPG that can indirectly influence the match score during the fixpoint computation. We name this method *Influence Nodes*.

#### 4.4.4 Implementation Notes on Matching

Matching is implemented on the same graph structures also used for IncGraph construction. PCG+s are built by wrapping nodes around existing node objects from IncGraphs and by stacking their labels. Thus, both underlying IncGraphs can still be navigated from any node in the PCG+. IPGs are built essentially by re-wiring individual aspects of PCG+s on the same objects. During matching, IPG objects are managed as changing object structures with live state. Consequently, there is no going back to previous states, once changes have been applied by matching algorithms. Resetting the initial state entirely requires a recalculation of both the PCG+ and IPG. This limitation is justified by *i<sup>3</sup>MAGE*'s principle to approach matching incrementally and without backtracking.

During matching, several indices and statistics are maintained on the IPG, which are in part exposed in a matching API. Adjustment of the IPG due to feedback is also possibly through the same API. Confirmation or rejection of individual matches can be applied by calling for a score change on a specific graph node with any of the three supported methods to either 1.0 or 0.0.

## 4.5 Mapping Generation

Mapping generation in *i<sup>3</sup>MAGE* can operate in two different modes. (1) Automatically, i.e., by providing a best-effort overall mapping that covers as much of the target ontology as possible. Or, (2) semi-automatically, i.e., based on manually confirmed suggestions.

### 4.5.1 Fully Automatic Mapping Selection

In fully automatic mode, *i<sup>3</sup>MAGE* exports mappings based on most likely IncMap correspondences calculated during the fixpoint computation. Correspondences are selected based on final scores after the last iteration of IncMap.

*i<sup>3</sup>MAGE* interprets correspondences either as 1 : 1 mappings or as  $n : 1$  mappings. Combinations of correspondences that lead to 1 :  $n$  interpretations are not systematically supported in automatic mode and hence the same holds for  $n : m$  mappings. This essentially means that, while *i<sup>3</sup>MAGE* can map several concepts to a single table, it cannot map a single ontology concept to several tables in automatic mode. What sounds like a harsh restriction at first is often less limiting in practice: quite often, the case where information about a single concept is stored in several tables of a relational database, this effectively represents a union of subclasses of that concept, and *i<sup>3</sup>MAGE* is still capable to map each of those subclasses separately if they are defined in the ontology. On the other side, disallowing 1 :  $n$  mappings in automatic generation tightens the search space and thus helps avoiding false positives.

Intuitively, following these rules, for each target side node (i.e., each node in the ontology IncGraph), one correspondence should be selected in automatic mode. We refer to this set of correspondences as the *target top-1* set of correspondences.

However, target top-1 correspondences may lead to significant inconsistencies, lowering the quality of the resulting mappings. For instance, the best match for a property will in some cases match its range class to a table other than the one chosen as best match for the class node of the range. While this might even be correct on occasion – either, because the range class match is the one, which is incorrect, or because keys in both matches define identical individuals – our general experience is that accepting those inconsistencies lowers mapping quality on average. Therefore, we do not select target top-1 correspondences but first choose target top-1 correspondences for classes only and then choose for properties from a restricted set that interprets domains and ranges consistently with the chosen class matches.

Finally, for each correspondence, one R2RML mapping rule is being generated. R2RML generation employs skolemization of URIs but is always deterministic, because our matches imply a complete semantical description of the target concept (through the underlying node from  $\text{IncGraph}(O)$ ) and a complete relational SPJ-access path on the source schema elements (through the underlying node from  $\text{IncGraph}(R)$ ). This is possible thanks to rich provenance added to IncGraphs at construction time.

### 4.5.2 Semi-automatic Mapping Selection

In semi-automatic mode, the selection process is driven by a user who confirms certain suggestions. For each suggestion, one R2RML mapping rule is immediately being generated. Over time, a set of mappings builds up, which is based on an arbitrary hand-curated selection of correspondences.

Both modes can be combined: a user may start to build mappings semi-automatically but at some point decide to complete all further mappings in fully automatic mode. As accepted suggestions in semi-automatic mode are also used as feedback for IncMap, automatic mapping in such a mixed approach is likely resulting in higher mapping accuracy.

### 4.5.3 Implementation Notes on Mapping Generation

Mapping generation relies on provenance information, which IncMap stores in its IncGraph nodes. Provenance contains aspects such as fully qualified database identifiers, ontology IRIs or additional context, such as a multi-hop join path underlying shortcut nodes. This essentially amounts to semantic knowledge of axioms on the target ontology side and to SPJ-type access in case of the relational database.

R2RML generation uses a commercial API for serialization.

## 4.6 Additional System Components

A number of additional supportive components are implemented in *i<sup>3</sup>MAGE* besides IncMap and besides the mapping generation component. Those additional modules support matching and mapping generation in productive setups.

### 4.6.1 Input Processing

While IncMap accepts its input through its own APIs, *i<sup>3</sup>MAGE* supports a range of reader APIs to connect to actual data sources.

A database reader connects to running relational DBMSs and reads their schema information and transforms those information for IncMap's API. With some limitations the database reader also accepts textual DDL as an alternative.

An OWL ontology reader is available to load target ontologies and transforms relevant axioms for IncMap's API. Ontologies can be imported in *i<sup>3</sup>MAGE* from file or from a named graph in a connected RDF database.

More advanced input is sometimes available in form of existing R2RML mappings. The mapping analyzer is designed as a component to read existing partial R2RML mappings and to derive correspondences from them as input for IncMap. Partial mappings could be the result of previous iterations running *i<sup>3</sup>MAGE*, or they could have been manually curated. Such extracted correspondences can be used in the same way as input from the feedback system.

Similarly, a query workload may be present as input. *i<sup>3</sup>MAGE* can analyze SPARQL queries for contained triple patterns, which are used in two different ways: firstly, to identify an area of specific interest in the target ontology and to ignore the rest of the ontology for suggestions, and secondly to derive concrete domain and range types.

### 4.6.2 Feedback System

Where IncMap provides suggestions (top-ranked matches) and solicits feedback from users (acceptance or rejection of suggestions), additional components need to mediate between native IncMap APIs and an external GUI that users can actually interact with.

*i<sup>3</sup>MAGE* implements a mapping suggester that exposes an API to external consumers, such as a GUI frontend. The API supports search and selection of suggestions by context (e.g., related to one particular class or table). The mapping suggester also translates suggestions into human readable form that can be displayed in textual or hypermedia format, including some explanation.

For receiving feedback, a collector component accepts feedback on suggested mappings via the same API as the mapping suggester. It then translates feedback back and forwards it to IncMap.

### 4.6.3 Implementation Notes on Additional Components

All components of *i<sup>3</sup>MAGE* are closely integrated in the overall system architecture, implemented in Java. The database reader is compatible with all JDBC-compliant RDBMSs that implement standard information schema.

The OWL reader is implemented using an external component, the OWL API [134]. In addition to standard OWL API functionality, the OWL reader implements a bridge to load ontologies directly from a Sesame OpenRDF database [137].<sup>6</sup>

R2RML import, analysis and export rely on a commercial Java API. Analysis is technically limited to mappings that use SQL tables or views for specification of their R2RML logical tables. So called R2RML views, i.e., ad hoc SQL views, cannot be analyzed due to a lack of SQL parsing in *i<sup>3</sup>MAGE*. Similarly, query analysis is constrained by the parser looking for basic triple patterns in the main WHERE clause of a SPARQL query, only (e.g., no information from SPARQL FILTERs or SPARQL SELECT sub queries can be considered).

All APIs in the feedback system are designed to be exposable as JSON APIs, although all applications so far interact with it directly through a Java interface.

---

<sup>6</sup><http://rdf4j.org>

## Chapter 5

# Benchmark Design and Evaluation

This chapter introduces the benchmark that we have developed to test and compare RDB2RDF mapping generating systems, and presents our evaluation of i<sup>3</sup>MAGE.

First, we motivate the need for a dedicated benchmark suite and give a general overview on our benchmark in Section 5.1. Next, we present our analysis of the different types of mapping challenges for RDB2RDF mapping generation in Section 5.2. Section 5.3 analyzes differences in mapping generation approaches that impact mapping generation, and thus also need to be considered for designing appropriate evaluation approaches. Then, Section 5.4 introduces our benchmark suite, called RODI, and discusses the evaluation procedure. Afterwards, Section 5.5 discusses implementation details that should help researchers and practitioners to understand, how other systems could be evaluated using RODI. Section 5.6 then presents our evaluation comparing i<sup>3</sup>MAGE to four other RDB2RDF systems, including a detailed discussion of observations. Finally, we summarize experimental findings for i<sup>3</sup>MAGE in Section 5.6.5.

### 5.1 Overview of Benchmark Design

Ontology-based data integration crucially depends on the quality of ontologies and mappings.

The quality of such generated RDB2RDF mappings is usually evaluated using self-designed and therefore potentially biased benchmarks, which make it difficult to compare results across systems. In practice, this is unsatisfactory since it does not provide enough ground to select an adequate mapping generation system in ontology-based data

integration projects. This limitation is evident in large scale industrial projects where support of automatic or semi-automatic systems is vital (e.g., [138, 139]). Thus, in order to ensure that ontology-based data integration can find its way into mainstream practice, there is a need for a generic and effective benchmark for reliable evaluation of the *quality* of computed mappings w.r.t. their utility under actual query workloads.

RODI, our mapping-quality benchmark for *Relational-to-Ontology Data Integration* scenarios, addresses this challenge.

The benchmark is composed of:

- A *framework* to test systems that generate mappings between relational schemata and OWL 2 ontologies. The RODI software package has been implemented and made available for public download under an open source license.<sup>1</sup>
- A *scoring function* to measure different facets of the quality of system-generated mappings.
- Different datasets and queries for benchmarking, which we call *benchmark scenarios*: RODI consists of 18 ontology-based data integration test scenarios from conference, geographical, and oil and gas domains. Scenarios are constituted of databases, ontologies, mappings, and queries to check expected results. Components of the scenarios are developed in such a way that they capture the key challenges of RDB2RDF mapping generation.
- An *extension mechanism* for adding further scenarios to the benchmark. Benchmark and scoring function are thus kept independent from benchmark scenarios, and the benchmark suite can be expanded easily to cover additional tests or application domains.

Using RODI one can evaluate the *quality* of RDB2RDF mappings produced by systems for ontology-based data integration from two perspectives: how good the mappings can translate between various particularities of relational schemata and ontologies, and how good they are from the query answering perspective.

To make this possible, RODI is designed as an end-to-end benchmark. That is, we consider systems that can produce mappings directly between relational databases and ontologies. Also, we evaluate mappings according to their utility for an actual query workload. Besides the benefit of testing mapping utility rather than theoretic properties, there are additional advantages of query workload-based evaluation that have recently been discussed in the literature [121, 140].

---

<sup>1</sup><https://github.com/chrp/rodi>

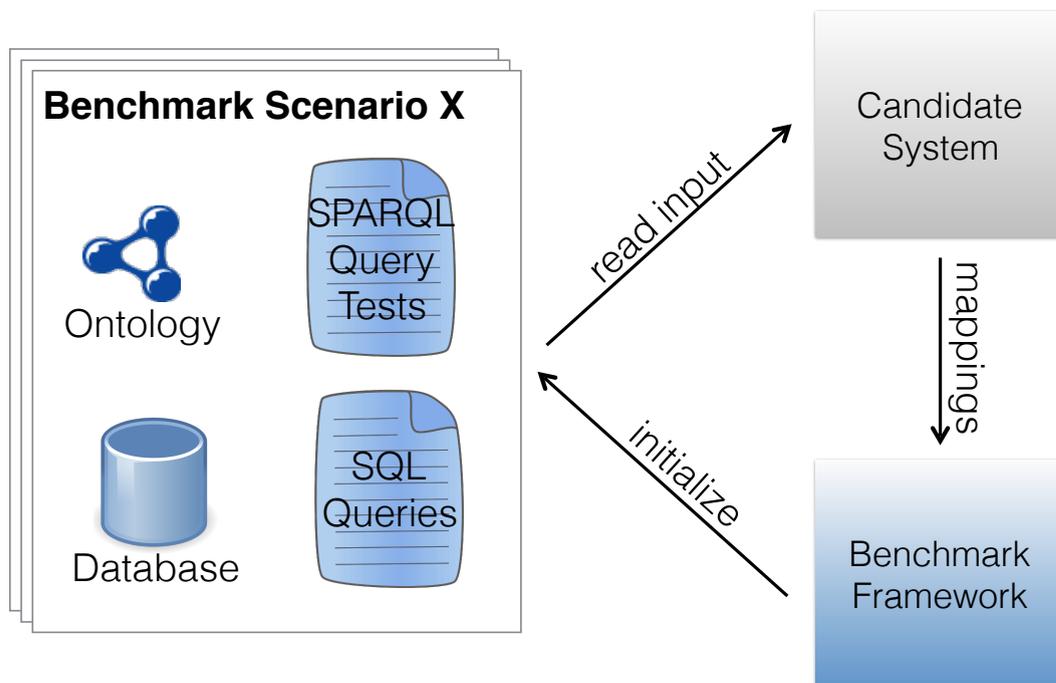


FIGURE 5.1: RODI benchmark overview

Figure 5.1 depicts the schematics of the resulting architecture: the benchmark includes a number of benchmark scenarios. Scenarios are initialized and setup for use by the framework. Candidate systems then read their input from the active scenario and produce mappings, which are evaluated again by our framework.

We have originally introduced RODI in [128]. Results presented in this work are based on a later version of the benchmark, which significantly extends earlier results:

- *Extended evaluation scenarios:* We provide 9 new evaluation scenarios that are important for testing mapping quality under real-world challenges such as high semantic heterogeneity or complex query workloads in different application domains.
- *Extended scope of the benchmark:* Besides fully automatic mapping generation, we can now also evaluate certain semi-automatic approaches and support several modes of evaluation.
- *Extended evaluation:* We evaluate an additional system, COMA++, which follows an approach that is closely related to i<sup>3</sup>MAGE. Although COMA++ is a much earlier system, not actively developed any longer, and although it was never designed specifically for RDB2RDF mappings, it represents a significant point of reference for i<sup>3</sup>MAGE’s core component IncMap as a baseline. This is because,

just like IncMap, COMA++ follows a generic graph matching approach and supports inter-model matching. Besides this addition, the discussion of evaluation results for all systems is significantly extended.

Besides, we have modified the benchmark scenarios to produce more specific individual scores rather than aggregated values for relevant categories of tests. We also extended the benchmark framework to allow detailed debugging of the results for each individual test. On that basis we can point to individual issues and bugs in several systems, some of which have already been addressed by the authors of the evaluated systems.

## 5.2 Integration Challenges

In the following we discuss our classification of different types of mapping challenges in RDB2RDF data integration scenarios. As a high-level classification, we use the standard classification for data integration described by Batini et al. [37]: naming conflicts, structural heterogeneity, and semantic heterogeneity. For each challenge, we describe the central issue of the problem and the main task faced by the mapping generation tools.

### 5.2.1 Naming Conflicts

Typically, relational database schemata and ontologies use different conventions to name their artifacts, even when they model the same domain and thus should use a similar terminology. While database schemata tend to use short identifiers for tables and attributes that often include technical artifacts (e.g. for tagging primary keys and for foreign keys), ontologies typically use long “speaking” names. Thus, the main challenge is to be able to find similar names despite the different naming patterns.

Other traditional differences include the use of plural vs. singular for class types, typically different tokenization schemes, etc. Those differences are not present in other cases of data integration (e.g., relational-to-relational or ontology alignment).

### 5.2.2 Structural Heterogeneity

The most important differences in RDB2RDF integration scenarios compared to other integration scenarios are structural heterogeneities. We discuss the different types of structural heterogeneity covered by RODI.

### 5.2.2.1 Type Conflicts

Relational schemata and ontologies represent the same artifacts by using different modeling constructs. While relational schemata use tables, attributes, and constraints, ontologies use modeling elements such as classes, data properties and object properties, restrictions, etc. Clearly, there exist direct (i.e., naive) mappings from relational schemata to ontologies for some of the elements (e.g., some classes immediately map to tables). However, most real-world relational schemata and corresponding ontologies cannot be related by any such naive mapping. This is because big differences exist in the way how the same concepts are modeled (i.e., type conflicts). Consequently, mapping rules need to be much more complex. One reason why these differences are so big is that relational schemata are often optimized towards a given workload (e.g., they are normalized for update-intensive workloads or denormalized for read-intensive workloads). Ontologies, on the other side, model a domain on the conceptual level. Another reason is that some modeling elements have no single direct translation (e.g., class hierarchies in ontologies can be mapped to relational schemata in different ways). In the following, we list the different type conflicts covered by RODI:

- *Normalization artifacts*: Often, properties that belong to a class in an ontology are spread over different tables in the relational schema as a consequence of normalization.
- *Denormalization artifacts*: For read-intensive workloads, tables are often denormalized. Thus, properties of different classes in the ontology might map to attributes in the same table.
- *Class hierarchies*: Ontologies typically make use of explicit class hierarchies. Relational models implement class hierarchies implicitly, typically using one of three different common modeling patterns (c.f., [28, Chap. 3]). In previous Section 4.2.7 we have discussed different relational patterns in detail. For class hierarchies, literature lists three different common patterns for relational databases:
  1. In one common variant the relational schema materializes several subclasses in the same table and uses additional attributes to indicate the subclass of each individual. Those additional attributes can take the shape of a numeric type column for disjoint subclasses and/or a combination of several type or role flags for non-disjoint subclasses. In this case, several classes need to be mapped to the same table and can be told apart only by secondary features in the data, such as the value in a type column. With this variant, mapping systems have to resolve  $n:1$  matches, i.e., they need to filter from one single table to extract information about different classes.

2. Another common way is to use one table per most specific class in the class hierarchy and to materialize the inherited attributes in each table separately. Thus, the same property of the ontology must be mapped to several tables. In this variant, mapping systems need to resolve 1: $n$  matches, i.e., build a union of information from several tables to retrieve entities for a single class.
3. A third variant uses one table for each class in the hierarchy, including the possibly abstract superclasses. Tables then use primary key-foreign key references to indicate the subclass relationship. This variant has a closer resemblance to ontology design patterns. However, it is also rarely used in practice, as it is more difficult to design, harder to query, impractical to update, and usually considered unnecessarily complex.

### 5.2.2.2 Key Conflicts

In ontologies and relational schemata, keys and references are represented differently. In the following, we list the different key conflicts covered by RODI:

- *Keys*: Keys in databases are usually implemented using primary keys and unique constraints. Keys may be composite, and in some cases partial keys of a table identify different related entities (e.g., denormalized tables on the relational side). Ontologies use IRIs as identifiers for individuals. Technically, OWL 2 also supports a notion of keys, but this feature is very rarely used.

Thus, the challenge is that integration tools must be able to generate mapping rules for creating IRIs for individuals from the correct choice of keys.

- *References*: A similar observation holds for references. While references are typically modeled as foreign keys in relational schemata, ontologies use object properties. Moreover, sometimes relational databases do not model foreign key constraints at all. In that case an integration tool must be able to derive references from the relational schema (e.g., based on the naming scheme and types or individuals).

### 5.2.2.3 Dependency Conflicts

These conflicts arise when a group of concepts are related among each other with different dependencies (i.e., 1:1, 1: $n$ ,  $n$ : $m$ ) in the relational schema and the ontology. Relational schemata may use foreign keys over attributes as constraints to explicitly model 1:1 and 1: $n$  relationships between different tables. They often model  $n$ : $m$  relationships using

an additional connecting table, which describes a *relationship relation*. Ontologies may model functionalities (i.e., functional properties or inverse functional properties), or they define cardinalities explicitly using cardinality restrictions. However, many ontologies do not make use of these restrictions and thus are often underspecified in this respect [141].

TABLE 5.1: Detailed list of specific structural mapping challenges. RDB patterns may correspond to some of the “guiding” ontology axioms. Specific difficulties explain particular hurdles in constructing mappings for those cases, which make the combinations unusually challenging.

#	Challenge type	RDB pattern	Guiding OWL axioms	Specific difficulty
(1)	<b>Normalization</b>	Weak entity table (depends on other table, e.g., in a part-of relationship)	owl:Class	JOIN to extract full IDs
(2)		1:n attribute	owl:DatatypeProperty	JOIN to relate attribute with entity ID
(3)		1:n relation	owl:ObjectProperty, owl:InverseFunctionalProperty	JOIN to relate entity IDs
(4)		n:m relation	owl:ObjectProperty	3-way JOIN to relate entity IDs
(5)		Indirect n:m relation (using additional intermediary tables)	owl:ObjectProperty	k-way JOIN to relate entity IDs
(6)	<b>Denormalization</b>	Correlated entities (in shared table)	owl:Class	Filter condition
(7)		Multi-value	owl:DatatypeProperty, owl:maxCardinality [ $>1$ ]	Handling of duplicate IDs
(8)	<b>Class hierarchies</b>	1:n property match	rdfs:subClassOf, owl:unionOf, owl:disjointWith	UNION to assemble redundant properties
(9)		n:1 class match with type column	rdfs:subClassOf, owl:unionOf	Filter condition
(10)		n:1 class match without type column	rdfs:subClassOf, owl:unionOf	JOIN condition as implicit filter
(11)	<b>Key conflicts</b>	Plain composite key	owl:Class, owl:hasKey	Technical handling
(12)		Composite key, n:1 class matching to partial keys	owl:Class, owl:hasKey, rdfs:subClassOf	Choice of correct partial keys
(13)		Missing key (e.g., no UNIQUE constraint on secondary key)	owl:Class, owl:hasKey	Choice of correct non-key attribute as ID
(14)		Missing reference (no foreign key where relevant relation exists)	owl:ObjectProperty, owl:DatatypeProperty	Unconstrained attributes as references
(15)	<b>Dependency conflicts</b>	1:n attribute	owl:FunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Misleading guiding axioms; possible restriction violations
(16)		1:n relation	owl:FunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Misleading guiding axioms; possible restriction violations
(17)		n:m relation	owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Misleading guiding axioms; possible restriction violations

Table 5.1 lists all specific testable RDB $\rightarrow$ RDF structural challenges that we have identified.

### 5.2.3 Semantic Heterogeneity

Semantic heterogeneity plays a highly important role for data integration in general. Therefore, we extensively test scenarios that bring significant semantic heterogeneity. This challenge is not specific to RDB2RDF data integration, but a property of data integration in general.

Besides the usual semantic differences between any two conceptual models of the same domain, three additional factors apply to RDB2RDF data integration:

- *Object-relational impedance mismatch*: An impedance mismatch caused by the object-relational gap, i.e., ontologies group information around entities (objects), while relational databases encode them in a series of values that are structured in relations.
- *CWA-OWA gap*: The impedance mismatch between the closed-world assumption (CWA) in databases and the open-world assumption (OWA) in ontologies.
- *Expressivity gap*: The difference in semantic expressiveness, i.e., databases may model some concepts or data explicitly, where they are derived logically in ontologies.

All of these factors are inherent to all RDB2RDF mapping problems.

## 5.3 Analysis of Mapping Approaches

Different mapping generation systems make different assumptions and implement different approaches. Thus, a benchmark needs to consider each approach appropriately. In the following, we first discuss the major differences regarding the availability of input. For instance, do we only have access to the ontology's T-Box axioms or also to some additional A-Box facts that could be used as data examples? Afterwards, we discuss the different approaches of implementing mapping processes and their effects on a benchmark, e.g., automatic vs. different forms of semi-automatic processes.

### 5.3.1 Differences in Availability and Relevance of Input

Different input may be available to an automatic mapping generator. In RDB2RDF data integration, the main difference between available inputs concerns the target ontology. The ontology could be specified entirely and in detail, or it could still be incomplete (or

even missing) when mapping construction starts. Moreover, other differences are also related to available input. For instance, data or a query workload could be available in addition to mere schema information on either side.

The case where both the relational database schema and the ontology are completely available could be motivated by different situations. For example, a company may wish to integrate a relational data source into an existing, mature, Semantic Web application. In this case, the target ontology would already be well defined and also be populated with some A-Box data. In addition, a SPARQL query workload could be known and could be available as additional input to a mapping generator.

On the other side, RDB2RDF data integration might be motivated by a large-scale industry data integration scenario (e.g., [139, 142]). In this scenario, the task at hand is to make complex and confusing schemata easier to understand for experts who write specialized queries. In this case, at the beginning no real ontology is given. At best there might be an initial, incomplete vocabulary. Mappings and ontology are basically being developed simultaneously over time. That is, no complete target ontology is available as input to a mapping generator.

Essentially, the different scenarios can all be distinguished by the following question: which information is available as input, besides the relational database? This can be a mix of an ontology's T-Box (or even just incomplete T-Box), A-Box data and an existing query workload in either SQL or SPARQL. Note, that we always assume that the relational source database is completely accessible (both schema and data), as this is a fundamental requirement, without which RDB2RDF data integration applications cannot reasonably be motivated. Besides the *availability* of input for mapping generation, there could be additional knowledge, about which parts of the input are even *relevant*. For instance, it may be clear that only parts of the ontology that are being used by a certain query workload need to be mapped. If so, this information could also be leveraged by the mapping generation system (e.g., by analyzing the query workload).

### 5.3.2 Differences in Mapping Process

Other differences can arise from the process in which mapping generation is approached. These can be either fully-automatic approaches or semi-automatic approaches. Truly semi-automatic approaches are usually iterative [84], as they consist of a sequence of mapping generation steps that get interrupted to allow human feedback, corrections, or other input. Their process is driven by the human perspective rather than by an automatic component. Since we want to better adjust our benchmark to the semi-automatic approaches, we first discuss different ways that are known for the semi-automatic case.

Heyvaert et al. [132] have recently identified four different ways for manual RDB2RDF mapping creation. Each of those directions inflicts a different interaction paradigm between the system and the user and thus solicits different forms of human input: users can edit mappings based on either the source or target definitions, they can drive the process by providing result examples or could theoretically even edit mappings irrespective of either the source or target in an abstract fashion. We have also earlier identified two fundamentally different user perspectives on mapping generation [131] that drive the process. They largely correspond to the first two ways described in [132].

Moreover, while some approaches consider manual corrections only at the end of the mapping process, more thoroughly semi-automatic approaches allow or even require such input during the process.

In terms of their potential evaluation, iterative approaches of this kind must be considered according to two additional characteristics: First, whether iterative human input is mandatory or generally optional. Second, whether input is only used to improve the mapping as such, or if the systems also exploit it as feedback for their next automated iteration. Systems that solicit input only optionally and do not use it as feedback can be evaluated like non-iterative systems on a fully automatic baseline without limitations. Systems with only optional input that *do* learn from the feedback (if provided), can still be evaluated on the same baseline but may not demonstrate their full potential. Where input is mandatory, systems need to be either steered by an actual human user or at least require simulated human input produced by an oracle.

Next, the kind of human input that a system can process makes a difference for evaluation settings. Most semi-automatic systems either provide suggestions that users can confirm or delete, or they allow users to adjust the mapping manually. An alternative approach is *mapping by example*, where users provide expected results. In addition, however, some systems may require complex or indirect interactions, or simply resort to more unusual forms of input that cannot easily be foreseen.

All the differences discussed before have an impact on how mapping generation systems need to be evaluated. Each mapping generation system is usually tied to one specific approach and does not allow for much freedom. We therefore decided that an end-to-end evaluation that allows the use of different types of input is best. Since semi-automatic approaches are becoming more and more relevant, we decided to support them using an automated oracle that simulates user input where possible.

## 5.4 RODI Benchmark Suite

In the following, we present the details of our RODI benchmark: we first give an overview, then we discuss the data sets (relational schemata and ontologies) that can be used, as well as the queries. Finally, we present our scoring function to evaluate the benchmark results.

### 5.4.1 Overview

Figure 5.2 gives an overview of the scenarios used in our benchmark. The benchmark ships with data sets from three different application domains: conferences, geodata and the oil & gas exploration domain. In its basic mode of operation, the benchmark provides one or more target ontologies for each of those domains (T-Box only) together with relational source databases for each ontology (schema and data). For some of the ontologies there are different variants of accompanying relational schemata that systematically vary the types of targeted mapping challenges.

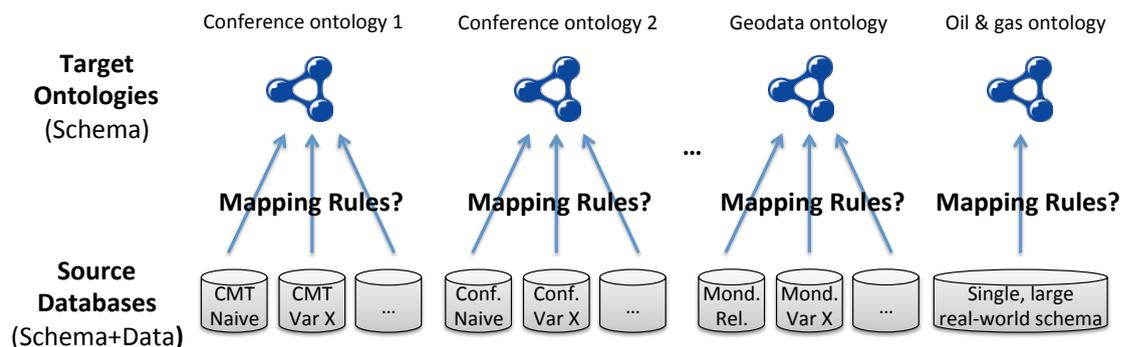


FIGURE 5.2: Overview of RODI benchmark scenarios

The benchmark asks systems to create mapping rules from the different source databases to their corresponding target ontologies. We call each such combination of a database and an ontology a *benchmark scenario*. For evaluation, we provide query pairs for each scenario to test a range of mapping challenges. Query pairs are evaluated against the instantiated ontology and the provided databases, respectively. Results are compared for each query pair and aggregated in the light of different mapping challenges using our scoring function.

While challenges that result from different naming or semantic heterogeneity are mostly covered by complete scenarios, we target structural challenges on a more fine-granular level of individual query tests with a dedicated score. Table 5.2 again lists individual

structural challenges and our coverage by dedicated tests. We cover all identified challenges. Most challenges (marked with a check in the table) are tested throughout a majority of scenarios. Missing constraints are modeled in only one dedicated scenario and are therefore only tested in that case. For dependency conflicts, we do not test with data, which would violate ontology restrictions, though. Instead, we only introduce the mismatch at schema level (i.e., misleading axioms), so all query tests still have exactly one correct solution.

TABLE 5.2: Coverage of structural challenges in default benchmark scenarios. Challenges marked with a check are tested throughout the majority of scenarios. 'Single scenario' marks challenges that could only be tested in a dedicated scenario. For dependency conflicts, we test only a part of the challenge (misleading axioms), but no restriction violations.

#	Challenge type	RDB pattern	Guiding OWL axioms	Covered
(1)	<b>Normalization</b>	Weak entity	owl:Class	✓
(2)		1:n attribute	owl:DatatypeProperty	✓
(3)		1:n relation	owl:ObjectProperty, owl:InverseFunctionalProperty	✓
(4)		n:m relation	owl:ObjectProperty	✓
(5)		Indirect n:m relation	owl:ObjectProperty	✓
(6)	<b>Denormalization</b>	Correlated entities	owl:Class	✓
(7)		Multi-value	owl:DatatypeProperty, owl:maxCardinality [ $>1$ ]	✓
(8)	<b>Class hierarchies</b>	1:n property match	rdfs:subClassOf, owl:unionOf, owl:disjointWith	✓
(9)		n:1 class match with type column	rdfs:subClassOf, owl:unionOf	✓
(10)		n:1 class match without type column	rdfs:subClassOf, owl:unionOf	✓
(11)	<b>Key conflicts</b>	Plain composite key	owl:Class, owl:hasKey	✓
(12)		Composite key, partial matching	owl:Class, owl:hasKey, rdfs:subClassOf	✓
(13)		Missing key	owl:Class, owl:hasKey	Single scenario
(14)		Missing reference	owl:ObjectProperty, owl:DatatypeProperty	Single scenario
(15)	<b>Dependency conflicts</b>	1:n attribute	owl:FunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Only misleading axioms
(16)		1:n relation	owl:FunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Only misleading axioms
(17)		n:m relation	owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:minCardinality [ $>1$ ], owl:maxCardinality [ $>1$ ], owl:cardinality [ $>1$ ]	Only misleading axioms

Multi-source integration can be tested as a sequence of different scenarios that share the same target ontology. We include specialized scenarios for such testing with the conference domain.

In order to be open for other data sets and different domains, our benchmark can be easily extended to include scenarios with real-world ontologies and databases. In our

initial version, we already provide one such extension from a real-world application of the oil and gas domain.

### 5.4.2 Data Sources and Scenarios

In the following, we present the data sources (i.e., ontologies and relational schemata) as well as the combinations used as integration scenarios for the benchmark in more details. RODI ships with scenarios based on data sources from three different application domains.

### 5.4.3 Conference Scenarios

As our primary domain for testing, we chose the conference domain: it is well understood, comprehensible even for non-domain experts but still complex enough for realistic testing and it has been successfully used as the domain of choice in other benchmarks before (e.g., by the OAEI [25]).

#### 5.4.3.1 Ontologies

The conference ontologies in this benchmark are provided by the Ontology Alignment Evaluation Initiative (OAEI) [25]. They were originally developed by the OntoFarm project [143]. We selected three particular ontologies (CMT, SIGKDD, CONFERENCE), based on a number of criteria: variation in size, the presence of functional cardinalities, the coverage of the domain, variations in modeling style, and the expressive power of the ontology language used. Different modeling styles result from the fact that each ontology was modeled by different people based on various views on the domain, e.g., they modeled it according to an existing conference management tool, expert insider knowledge, or according to a conference website. To cover our mapping challenges (Section 5.2), we selectively modified the ontologies (e.g., we included labels to add interesting lexical matching challenges). In SIGKDD, we have fixed a total of seven inconsistencies that we discovered in this ontology as follows: (1) We selectively added annotations like labels and comments, as these can help to identify correspondences lexically; (2) we added a few additional datatype properties where they were scarce, as they test other mapping challenges than just classes and object properties; and (3), we fixed a total of seven inconsistencies that we discovered in SIGKDD when adding A-Box facts (e.g., each place with a zip code automatically became a sponsor, who were modeled as a subclass of persons).

### 5.4.3.2 Relational Schemata

We synthetically derived different relational schemata for each of the ontologies, focusing on different mapping challenges. We provide benchmark scenarios as combinations of those derived schemata with either their ontologies of origin, or, for more advanced testing, paired with any of the other ontologies. First, for each ontology we derived a relational schema that can be mapped to the ontology using a naive mapping as described in [135]. The algorithm works by deriving an ER model from an OWL DL ontology. It then translates this ER model into a relational schema according to text book rules (e.g., [28]). In this work, we extended the algorithm to cover the full range of expected relational design patterns. In particular, the previous version did cover only one out of the above-mentioned three design patterns to translate class hierarchies into relational tables. Additionally, we extended this algorithm to consider ontology instance data to derive more proper functionalities (rather than just looking at the T-Box as the previous algorithms do). Otherwise, the generated naive relational schemata would have contained an unrealistically high number of  $n:m$ -relationship tables. The naively translated schemata of the algorithm are guaranteed to be in fourth normal form (4NF), fulfilling normalization requirements of standard design practices. Thus, the naive schemata already include various normalization artifacts as mapping challenges.

From each naively translated schema, we systematically created different variants by introducing different aspects on how a real-world schema may differ from a naive translation and thus to test different mapping challenges:

1. *Adjusted Naming*: As described in Section 5.2.1, ontology designers typically consider other naming schemes than database architects do, even when implementing the same (verbal) specification. Those differences include longer vs. shorter names, “speaking” prefixes, human-readable property IRIs vs. technical abbreviations (e.g., “hasRole” vs. ”RID”), camel case vs. underscore tokenization, preferred use of singular vs. plural, and others. For each naively translated schema we automatically generate a variant with identifier names changed accordingly.
2. *Restructured Hierarchies*: The most critical structural challenge in terms of difficulty comes with different relational design patterns to model class hierarchies more or less implicitly. As we have discussed in Section 5.2.2, these changes introduce significant structural dissimilarities between source and target. We automatically derive variants of all naively translated schemata, where different hierarchy design patterns are presented. The choice of design pattern in each case is algorithmically determined on a “best fit” approach considering the number of specific and shared (inherited) attributes for each of the classes.

3. *Combined Case*: In the real world, both of the previous cases (i.e., adjusted naming and hierarchies) would usually apply at the same time. To find out how tools cope with such a situation, we also built scenarios where both are combined.
4. *Removing Foreign Keys*: Although it is considered as bad style, databases without foreign keys are not uncommon in real-world applications. This can be a result from either lazy design or come with legacy applications (e.g., one popular open source DBMS introduced plugin-free support for foreign keys less than five years ago). The mapping challenge is that mapping tools must find the join paths to connect tables of different entities. Additionally, they sometimes even need to guess a join path for reading attributes of the same entity, if its data is split over several tables as a consequence of normalization. Therefore, we have created one dedicated scenario to test this challenge with the CONFERENCE ontology and based it on the schema variant with restructured hierarchies.
5. *Partial Denormalization*: In many cases, schemata get partially denormalized to optimize for a certain read-mostly workload. Denormalization essentially means that correlated (yet separated) information is jointly stored in the same table and partially redundant. We provide one such scenario for the CMT ontology. As denormalization requires conscious design choices, this schema is the only one that we had to hand-craft. It is based on the variant with restructured hierarchies.

In some cases, data transformations may also be required for a mapping to fully work as expected. A significant number of fundamentally different transformation types needs to be considered, each adding complexity in a different way. These comprise translations between different representations of date and time (e.g., a dedicated date type versus Epoch time stamps), simple numeric unit transformations (e.g., MB vs. GB), unit transformations requiring more complex formulae (e.g., degrees Celsius vs. Fahrenheit), string based data cleansing (e.g., removing trailing white space), string compositions (e.g., concatenating a first and last name), more complex string modifications (e.g., breaking up a string based on a learned regular expression), table based name translations (e.g., replacing names using a thesaurus), noise removal (e.g., ignoring erroneous tuples), etc.

While our extension mechanism (see Section 5.4.6) is suited to even add dedicated scenarios for testing such conversions, we excluded them from our default benchmark for mere practical reasons: (1) To the best of our knowledge, they play no role in any RDB2RDF mapping generation system to date, so there is little practical relevance as of now. And (2), not all of the different transformation types typically co-occur in the same application domain, and it would be hard to incorporate them into our conference domain scenario in appropriate variety without making the scenario less realistic.

TABLE 5.3: Basic scenario variants

	CMT	CONFERENCE	SIGKDD
Naive	(✓)	(✓)	(✓)
Adjusted Naming	✓	✓	✓
Restructured Hierarchies	✓	✓	✓
Combined Case	(✓)	(✓)	✓
Missing FKs	-	✓	-
Denormalized	✓	-	-

### 5.4.3.3 Scenario Variants

For each of our three main ontologies, CMT, CONFERENCE, and SIGKDD, the benchmark includes five scenarios, each with a different variant of the database schema (discussed before). Table 5.3 lists the different versions.

As discussed before, *Naive* closely mimics the structure of the original ontology, but the schemata are normalized and thus the scenario contains the challenge of normalization artifacts. *Adjusted Naming* adds the naming conflicts as discussed before. *Restructured hierarchies* tests the critical structural challenge of different relational patterns to model class hierarchies, which, among others, subsumes the challenge to correctly build  $n:1$  mappings between classes and tables. In the *Combined Case*, renamed, restructured hierarchies are employed and their effects are tested in combination. This is a more advanced test case. A special challenge arises from databases with no (or few) foreign key constraints (*Missing FKs*). In such a scenario, mapping tools must guess the join paths to connect tables that correspond to different entity types. The technical mapping challenge arising from *Denormalized* schemata consists in identifying the correct *partial* key for each of those correlated entities, and in identifying, which attributes and relations belong to which of the types.

To keep the number of scenarios small for the default setup, we differentiate between default scenarios and non-default scenarios. We excluded scenarios with the most trivial schema versions. In addition, we did limit the number of combinations for the most complex schema versions by including only one of each type as a default scenario. While the default scenarios are mandatory to cover all mapping challenges, the non-default scenarios are optional (i.e., users could decide to run them in order to gain additional insights). Non-default scenarios are put in parentheses in Table 5.3. However, they are not supposed to be executed in a default run of the benchmark.

Similarly, we include scenarios that require mappings of schemata to one of the other ontologies (e.g., mapping a CMT database schema variant to the SIGKDD ontology).

They represent more advanced data integration scenarios and are part of default scenarios.

#### 5.4.3.4 Data

We provide data to fill both the databases and ontologies. Conference ontologies are originally provided as T-Boxes, only, i.e., no A-Box. We first generate data as A-Box facts for the different ontologies, and then translate them into the corresponding relational data. Transformation of data follows the same process as translating the T-Box. For evaluation, data is only needed in the relational databases, so, generating ontology A-Boxes would not even be necessary. However, this procedure simplifies data generation, since all databases can be automatically derived from the given ontologies as described before.

Our conference data generator deterministically produces a scalable amount of synthetic facts around key concepts in the ontologies, such as conferences, papers, authors, reviewers, and others. In total, we generate data for 23 classes, 66 object properties (including inverse properties) and 11 datatype properties (some of which apply to several classes). However, not all of those concepts and properties are supported by every ontology. For each ontology, we only generate facts for the subset of classes and properties that have an equivalent in the relational schema in question.

#### 5.4.3.5 Queries

We test each integration scenario with a series of *query pairs*, consisting of semantically equivalent queries against the instantiated ontology and the provided databases, respectively.

Query pairs are manually curated and designed to test different mapping challenges. To this end, all query pairs are tagged with categories, relating them to different mapping challenges. All scenarios draw on the same pool of 56 query pairs, accordingly translated for each ontology and schema. However, the same query may face different challenges in different scenarios, e.g., a simple 1:1 mapping between a class and table in a naive scenario can turn into a complicated  $n:1$  mapping problem in a scenario with restructured hierarchies. Also, not all query pairs are applicable on all ontologies (and thus, on their derived schemata).

Query pairs are grouped into three basic categories to test the correct mapping of *class instances*, instantiations of *datatype properties* and *object properties*, respectively. Additional categories relate queries to  $n:1$  and  $n:m$  mapping problems or prolonged property

join paths resulting from normalization artifacts. A specific category exists for the denormalization challenge.

#### 5.4.4 Geodata Domain – Mondial Scenarios

As a second application domain, RODI ships scenarios in the domain of geographical data.

The Mondial database is a manually curated database containing information about countries, cities, organizations, as well as about geographic features such as waters (with subclasses lakes, rivers, and seas), mountains, and islands. It has been designed as a medium-sized case study for several scientific aspects and data models [144].

Based on Mondial, we have developed a number of benchmark scenarios. First, there is a scenario based on the original relational database, which features a wide range of relational modeling patterns, and the Mondial OWL ontology. In addition, we have added a series of further scenarios with synthetically modified variants of the database to focus on the effect of specific different relational modeling patterns. This is similar to the different variants produced in the conference domain. To keep the number of tested scenarios at bay, we do not consider those additional synthetic variants as part of the default benchmark. Instead, we recommend to only test the main Mondial scenario with others being available as optional tests to dig deeper into specific behavioral patterns in this domain.

In all scenarios, we use a query workload that mainly approximates real-world explorative queries on the data, although limited to queries of low or medium complexity. Still, those queries typically co-relate more than one concept or require several attributes to be correctly mapped at the same time in order to return any correct results. The degree of difficulty in Mondial scenarios is therefore generally higher than the one of our scenarios in the conference domain.

#### 5.4.5 Oil & Gas Domain – NPD FactPages Scenarios

Finally, we include an example of an actual real-world database and ontology in the oil and gas domain: The Norwegian Petroleum Directorate (NPD) FactPages [14]. Our test set contains a small relational database (approximately 40 MB), but with a relatively complex structure (70 tables, around 1000 total columns and approximately 100 foreign keys), and an ontology covering the domain of the database. The database is constructed from a publicly available dataset containing reference data about past and ongoing activities in the Norwegian petroleum industry, such as oil and gas production and

exploration. The corresponding ontology contains around 300 classes and more than 300 different properties.

With this pair of a database and an ontology, we have constructed two scenarios that feature a different series of tests on the data: first, there are queries that are built from information needs collected from real users of the FactPages and cover large parts of the dataset. Those queries are highly complex compared to the ones in other scenarios and require a significant number of schema elements to be correctly mapped at the same time to bear any results. We have collected 17 such queries in scenario *npd\_user\_tests*. And second, we have generated a large number of small, atomic query tests for baseline testing. These are similar to the ones used with the conference domain, i.e., they test for individual classes or properties to be correctly mapped. A total of 439 such queries have been compiled in scenario *npd\_atomic\_tests* to cover all of the non-empty fields in our sample database.

A specific feature resulting from the structure of the FactPages database and ontology is a high number of 1:*n* matches, i.e., concepts or properties in the ontology that require a UNION over several relations to return complete results. 1:*n* matches as a structural feature can therefore best be tested in the *npd\_atomic\_tests* scenario.

#### 5.4.6 Extension Scenarios

Our benchmark suite is designed to be extensible, i.e., additional scenarios can be easily added. The primary aim of supporting such extensions is to allow domain-specific, real-world mapping challenges to be tested alongside our default scenarios. Extension scenarios can be added by users of our benchmark without any programming efforts. Also, creating and adding scenarios are described in the user documentation of the RODI benchmark suite.

#### 5.4.7 Evaluation Criteria – Scoring Function

It is our aim to measure the practical usefulness of mappings. We are therefore interested in the utility of query results, rather than comparing mappings directly to a reference mapping set or than measuring precision and recall on all elements of the schemata. This is important because a number of different mappings might effectively produce the same data w.r.t. a specific input database. Also, the mere number of facts is no indicator of their semantic importance for answering queries (e.g., the overall number of conferences is much smaller than the number of paper submissions, yet they are at least as important in a query on information about the same papers). In addition, in many

cases only a subset of the information is relevant in practice, and we define our queries on a meaningful subset of information needs.

We therefore observe a score that reflects utility of the mappings with relation to our query tests as our main measure. Intuitively, this score reports the percentage of successful queries for each scenario.

However, in a number of cases, queries may return correct but incomplete results, or could return a mix of correct and incorrect results. In these cases, we consider per-query accuracy by means of a local per-query F-measure. Technically, our reported overall score for each scenario is the average of F-measures for each query test, rather than a simple percentile of successful queries. To calculate these per-query F-measures, we also need to consider query results that contain IRIs.

Apparently, different mapping generators will generate different IRIs for the same entities, e.g., by choosing different prefixes. F-measures for query results containing IRIs are therefore w.r.t. the degree to which they satisfy *structural equivalence* with a reference result. For practical reasons, we use query results on the original, underlying SQL databases as technical reference during evaluation. Structural equivalence effectively means that if same-as links were established appropriately, then both results would be semantically identical.

Formally, we define precision and recall locally for each individual test (i.e., for each query pair) and use a simple scoring function to calculate averages for different subsets of tests, i.e., for tests relating to a specific mapping challenge. Note, that it is still possible with this approach to evaluate all produced data by including a query like `CONSTRUCT WHERE {?s ?p ?o}`.

Unfortunately, precision and recall cannot be measured by naively comparing results of query pairs tuple by tuple, as different mappings typically generate different IRIs to denote the same entities. Instead, we define an equivalence measure between mappings that is agnostic of entity IRIs called *mapping equivalence*.

In the following, we define *mapping equivalence* based on a more general equivalence of query results (i.e., tuple sets):

**Definition 5.1** (Structural Tuple Set Equivalence). Let  $V = IRI \cup Lit \cup Blank$  be the set of all IRIs, literals and blank nodes,  $T = V \times \dots \times V$  the set of all  $n$ -tuples of  $V$ . Then two tuple sets  $t_1, t_2 \in \mathcal{P}(T)$  are *structurally equivalent* if an isomorphism  $\phi : (IRI \cap t_1) \rightarrow (IRI \cap t_2)$ .

For instance,  $\{(\text{urn:p-1}, \text{'John Doe'})\}$  and  $\{(\text{http://my\#john}, \text{'John Doe'})\}$  are structurally equivalent. On this basis, we can easily define the equivalence of query results w.r.t. a mapping target ontology:

**Definition 5.2** (Tuple Set Equivalence w.r.t. Ontology ( $\sim_O$ )). Let  $O$  be a target ontology of a mapping,  $I \subset IRI$  the set of IRIs used in  $O$  and  $t_1, t_2 \in \mathcal{P}(T)$  result sets of queries  $q_1$  and  $q_2$  evaluated on a superset of  $O$  (i.e., over  $O$  plus A-Box facts added by a mapping).

Then,  $t_1 \sim_O t_2$  (are structurally equivalent w.r.t.  $O$ ) iff  $t_1$  and  $t_2$  are structurally equivalent and  $\forall i \in I : \phi(i) = i$

For instance,  $\{(\text{urn:p-1}, \text{'John Doe'})\}$  and  $\{(\text{http://my\#john}, \text{'John Doe'})\}$  are structurally equivalent, *iff*  $\text{http://my\#john}$  is *not* already defined in the target ontology. Finally, we can define mapping equivalence:

**Definition 5.3** (Mapping Equivalence w.r.t.  $O$ ). Let  $m_1, m_2 \in M$  be mappings from relational databases  $R_1, R_2$  to a target ontology  $O$ , and  $Q$  be the set of queries applicable on  $O$ .

Then,  $m_1, m_2$  are equivalent w.r.t. target ontology  $O$  iff:  $\forall q \in Q : q(O \cup m_1(R_1)) \sim_O q(O \cup m_2(R_2))$ .

In other words, two mappings are equivalent w.r.t. a target ontology if every possible query will produce structurally equivalent result sets w.r.t. that ontology when it runs on data generated by one of the mappings versus the other. In practice, we evaluate against a specified subset of all possible queries covering interesting parts of the target ontology reasonably well.

We observe precision and recall locally on each query test, i.e., based on how many of the result tuples of each query are structurally equivalent to a reference query result set. Formally:

**Definition 5.4** (Precision and Recall under Structural Equivalence). Let  $t_r \in \mathcal{P}(T)$  be a reference tuple set,  $t_t \in \mathcal{P}(T)$  a test tuple set and  $t_{rsub}, t_{tsub} \in \mathcal{P}(T)$  be maximal subsets of  $t_r$  and  $t_t$ , s.t.,  $t_{rsub} \sim_O t_{tsub}$ .

Then the precision of the test set  $t_t$  is  $P = \frac{|t_{tsub}|}{|t_t|}$  and recall is  $R = \frac{|t_{rsub}|}{|t_r|}$ .

Table 5.4 shows an example with a query test that asks for the names of all authors. Result set 1 is structurally equivalent to the reference result set, i.e., it has found all authors and did not return anything else, so both precision and recall are 1.0. Result set

TABLE 5.4: Example results from a query pair asking for author names  
(e.g., SQL: SELECT name FROM persons WHERE person\_type=2;  
SPARQL: SELECT ?name WHERE ?p a :Author; foaf:name ?name)

(A) Reference	(B) Result 1	(C) Result 2	(D) Result 3
John	Jane	John	Jane
Jane	John		John
			James

2 is equivalent with only a subset of the reference result (e.g., it did not include those authors who are also reviewers). Here, precision is still 1.0, but recall is only 0.5. In case of result set 3, all expected authors are included, but also another person, James. Here, precision is 0.66 but recall is 1.0.

To aggregate results of individual query pairs, a scoring function calculates the averages of per query numbers for each scenario and for each challenge category. For instance, we calculate averages of all queries testing 1:*n* mappings. Thus, for each scenario there is a number of scores that rate performance on different technical challenges. Also, the benchmark can log detailed per-query output for debugging purposes.

#### 5.4.8 System Requirements

With RODI, we can test mapping generators that work in either one or two stages: that is, they either directly map data from the relational source database to the target ontology in one stage (e.g., i<sup>3</sup>MAGE or also COMA++ [24]). Or, they bootstrap their own ontology, which they use as an intermediate mapping target. In this case, to get to the full end-to-end mappings that we can test, the intermediate ontology and the actual target ontology should be integrated via ontology alignment in a second stage. Two-stage systems may either include a dedicated ontology alignment stage (e.g., [19]) or they deliver the first stage only ([21, 79]). In the latter case, RODI can step in to fill the missing second stage with a standard ontology alignment setup.

Our tests check the accuracy of SPARQL query results. Queries ask for individuals of a certain type (or their aggregates), properties correlating them, associated values and combinations thereof, sometimes also using additional SPARQL language features such as filters to narrow down the result set. This means that mapped data will be deemed correct if it contains correct RDF triples for all tested cases. For entities, this means that systems need to construct one correctly typed IRI for each entity of a certain type. For object properties, they need to construct triples to correctly relate those typed IRIs, and for datatype properties, they need to assign the correct literal values to each of the entity IRIs using the right predicates. Systems do therefore not strictly need

to understand or to produce any OWL axioms in the target ontology. However, our target ontologies are in OWL 2, using different degrees of expressiveness. Axioms in the target ontology can be important as *guidance* to identify suitable correspondences for one-stage systems. Similarly, if two-stage systems *construct* expressive axioms in their intermediate ontology, this may guide the second stage of ontology alignment. For instance, if a predicate is known to be an object property in the target ontology, results will suffer if a mapping generation tool assigns literal values using this property. Also, if a property is known to be functional it might be a better match for a  $n:1$  relation than a non-functional property would be.

## 5.5 Framework Implementation

In this section, we discuss some implementation details in order to guide researchers and practitioners to include their system in our benchmarking suite.

### 5.5.1 Architecture of the Benchmarking Suite

Figure 5.3 depicts the overall architecture of our benchmarking suite. The framework requires upfront initialization *per scenario*. Artifacts generated or provided during initialization are depicted blue in the figure. After initialization, a mapping tool can access the database (directly or via the framework's API) and target ontology (via the Sesame OpenRDF API [137] or using SPARQL, or serialized as an RDF file). Finally, it submits generated R2RML<sup>2</sup> mappings in a special folder on the file system, so evaluation can be triggered. As an alternative, mapping tools could also execute mappings themselves and submit final mapped data instead of R2RML. This would be the preferred procedure for tools that do not support R2RML but other mapping languages. More generally, mapping tools that cannot comply with the assisted benchmark workflow can always trigger individual aspects of initialization or evaluation separately.

### 5.5.2 Details on the Evaluation Phase

Unless a mapping system under evaluation decides to skip individual steps, i.e., to implement them independently, in the evaluation phase, the benchmark suite will:

1. Read submitted R2RML mappings and execute them on the database.

---

<sup>2</sup><http://www.w3.org/TR/r2rml/>

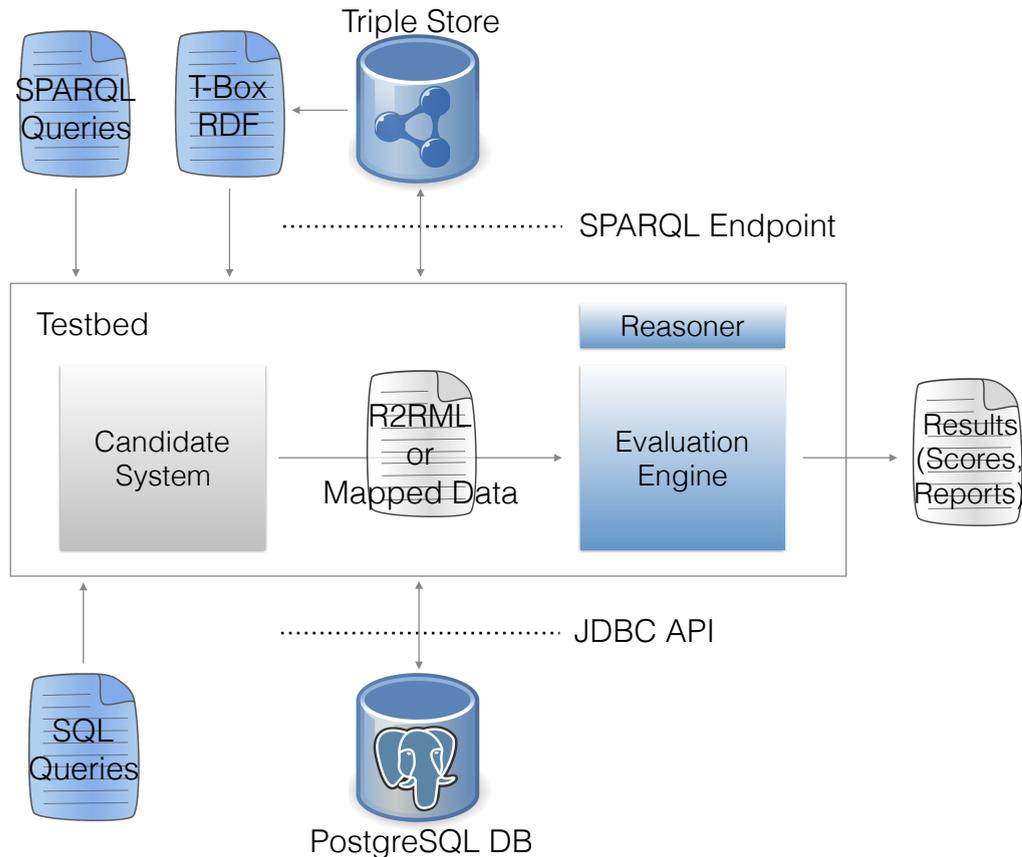


FIGURE 5.3: RODI framework architecture

2. Materialize the resulting A-Box facts in a Sesame RDF repository together with the target ontology (T-Box).
3. Optionally apply reasoning through an external OWL API [134] compatible reasoner to infer additional facts that may be requested for evaluation.
4. Evaluate all query pairs of the scenario on the repository and relational database.
5. Produce a detailed evaluation report.

We evaluate query results as described in Section 5.4.7 by attempting to construct an isomorphism  $\phi$  to transform query result sets into reference results. Technically, we use the results of the SQL queries from query pairs to calculate the reference result set. For each SQL query in a query pair, we flag attributes that together serve as keys, so keys can be matched with IRIs rather than with literal values. Obviously, keys and IRIs need to match only on the count of being the same unique value wherever they appear, while literal values need to be exact matches.

For constructing  $\phi$ , we first index all individual IRIs (i.e., IRIs that identify instances of some class) in the query result. Next, we build a corresponding index for keys in the reference set. For both sets we determine binding dependencies across tuples (i.e., re-occurrences of the same IRI or key in different tuples). As a next step, we narrow down match candidates to tuples where all corresponding literal values are exact matches. Finally, we match complete result tuples with reference tuples, i.e., we also check for viable correspondences between keys and IRIs. As discussed, the criterion for a viable match between a key and an IRI is that for each occurrence of this particular key and of this particular IRI in any of the tuples, both need to be matched with the same partner. This last step corresponds to identifying a maximal common subgraph (MCS) between the dependency graphs of tuples on both sides, i.e., it corresponds to the MCS-isomorphism problem. For efficiency reasons, we approximate the MCS if dependency graphs contain transitive dependencies, breaking them down to fully connected subgraphs. However, it is usually possible to formulate query results to not contain any such transitive dependencies by avoiding inter-dependent IRIs in SPARQL SELECT results in favor of a set of significant literals describing them. All queries shipped with this benchmark are free of transitive dependencies, hence the algorithm is accurate for all delivered scenarios.

Finally, we count tuples that could not be matched in the result and reference set, respectively. Precision is then calculated as  $\frac{|res| - |unmatched(res)|}{|res|}$  and recall as  $\frac{|ref| - |unmatched(ref)|}{|ref|}$ . Aggregated numbers are calculated per query pair category as the averages of precision and recall of all queries in each category.

## 5.6 Benchmark Results

We have performed an in-depth analysis comparing i<sup>3</sup>MAGE with a range of other systems using RODI.

### 5.6.1 Evaluated Systems

We perform experiments with three different configurations of IncMap running inside i<sup>3</sup>MAGE:

1. *IncMap Basic* is the latest version of IncMap without specialized reasoning pattern support. This is to demonstrate how IncMap can cope as a general-purpose inter-model matching tool, only with graph structures optimizations for the specific models.

2. *IncMap Complete* is the full latest version of IncMap, including special rule-based reasoning and pattern support during graph construction.
3. *IncMap QW* uses the same system configuration as IncMap Complete, but allows the system to peek into the SPARQL side of the query workload (QW). As one of the interaction paradigms of i<sup>3</sup>MAGE is to be query-driven, the system uses queries to steer matching into areas of interest, and to reinforce connections of particular relevance to the queries.

As competing systems we evaluate the current contender in the automatic mapping generation segment, *BootOX* [19]. Also, we evaluate against more general-purpose mapping generators that we combine with ontology alignment to measure in the benchmark (*-ontop-* [79] and *MIRROR* [21]). In addition, we test a much earlier, yet state-of-the-art system in inter-model matching, *COMA++* [24].

1. *BootOX* is based on the approach called *direct mapping* by the W3C [109]: every table in the database (except for those representing *n:m* relationships) is mapped to one class in the ontology; every data attribute is mapped to one data property; and every foreign key to one object property. Explicit and implicit database constraints from the schema are also used to enrich the bootstrapped ontology with axioms about the classes and properties from these direct mappings. Afterwards, *BootOX* performs an alignment with the target ontology using the LogMap system [58, 145, 146].

*BootOX* is a current contender in systems designed specifically for automatic RDB $\rightarrow$ RDF mapping generation with end-to-end capabilities and thus a direct competitor of i<sup>3</sup>MAGE.

2. *MIRROR* is a tool for generating an ontology and R2RML direct mappings automatically from a relational schema. It has been implemented as a module of the RDB $\rightarrow$ RDF engine morph-RDB [147]. *MIRROR* is specialized in extracting sophisticated design patterns from databases. Its output is oblivious of the required target ontology, though, so we perform post-processing with the ontology alignment tool LogMap [58].

Like *BootOX*, *MIRROR* is a current contender in RDB $\rightarrow$ RDF mapping generation. As discussed, it does not support end-to-end mapping generation on its own but does require post-processing using ontology alignment. This is because it has been designed and optimized for slightly different use cases.

3. The *-ontop- Protege Plugin* is a mapping generator developed for *ontop* [79]. *-ontop-* is a full-fledged query rewriting system with limited ontology and mapping bootstrapping capabilities.

-ontop- can be seen as a rather naive baseline case, which is very close to the W3C's direct mapping [109]. Like MIRROR, it has been built for a slightly different use case and requires additional ontology alignment to produce end-to-end mappings.

4. *COMA++* has been a contender in the field of schema matching for several years already; it is still widely considered state of the art. In contrast to other systems from the same era, *COMA++* is built explicitly also for inter-model matching. To evaluate the system, we had to perform a translation of its output correspondences into modern R2RML.

*COMA++* is an important baseline for *i<sup>3</sup>MAGE* as they both apply a matching approach that is mostly based on structural graph matching, and *COMA++* also supports RDB2RDF matching. Despite the fact that *COMA++* was never *specifically* designed for RDB2RDF matching and does therefore not implement any tuning for this specific case, it features a generic multi-model matching approach and, to the best of our knowledge, is the only established system based on graph matching to support RDB2RDF.

## 5.6.2 Experimental Setup

### 5.6.2.1 Automatic Experiments

For all systems, we conduct default experiments from the RODI benchmark suite as described in Section 5.4. This includes a selection of nine prototypical scenarios from the conference domain, one from the geodata domain and two from the oil & gas domain, as well as five different cross-matching scenarios. For all of these main experiments, we observe and report overall RODI scores as well as specific selected scores in individual categories.

These experiments are suited to demonstrate the overall capabilities of *i<sup>3</sup>MAGE* in different situations and vis-a-vis other systems on an easily comparable basis of fully automatic matching and mapping generation. They also demonstrate differences between the distinct versions and setups of IncMap and highlight the impact of the different core features used in these versions.

### 5.6.2.2 Incremental Experiments

In addition, we perform incremental, semi-automatic experiments on a subset of scenarios specifically for *i<sup>3</sup>MAGE*. A direct comparison with competing systems is not

possible, because none of the other systems tested supports a semi-automatic incremental approach that is comparable with i<sup>3</sup>MAGE. Also, other published systems that *do* support semi-automatic mapping generation, most notably Karma [18], would not be directly comparable. In Karma’s case this is because it solicits completely different types of human input but also requires a different integration workflow, with (semi-)automatic support kicking in only in cases of multi-source data integration, while i<sup>3</sup>MAGE does offer automatic mapping support for single-source mappings. We therefore did not include such systems in the evaluation.

With IncMap, we could simulate human feedback by responding to suggestions by taking a response from the benchmark that indicates changes in mapping quality. We assume an interface that presents users with a short list of strictly alternative mapping suggestions and asks them to either point out the correct one, or mark them as inadequate. Mapping suggestions are kept minimal, e.g., to map individuals of one particular class in the ontology, values for one datatype property in the context of one particular class. To simulate a human user, we use the benchmark suite as an oracle, submitting the different partial mapping suggestions separately and observing the score. We then report overall benchmark scores on mapping quality after  $k$  succinct interactions, i.e., a “score @ $k$  interactions”. This corresponds to the evaluation approach suggested in [148]. We manually fixed a randomly chosen order of query tasks (i.e., partial mappings) to test in i<sup>3</sup>MAGE.

This line of experiments is designed to demonstrate the feasibility and utility of i<sup>3</sup>MAGE’s interactive and incremental mode of operation.

### 5.6.3 Automatic Experiments: Results

Tables 5.5–5.10 show overall scores for all systems on all default scenarios from the RODI benchmark.

#### 5.6.3.1 Overall Conference Domain Results

For most scenarios, i<sup>3</sup>MAGE in at least one of the different IncMap setups outperforms all other systems with varying margins. Between the different versions of IncMap, a positive impact of core features activated in the more advanced versions could be generally observed. In almost all cases, i<sup>3</sup>MAGE performs better with IncMap Complete (i.e., enabling custom reasoning rules and advanced patterns) than it does with IncMap Basic. Also, leveraging the query workload (IncMap QW) has an additional positive effect, although its impact is moderate in most cases. It is important to note that no

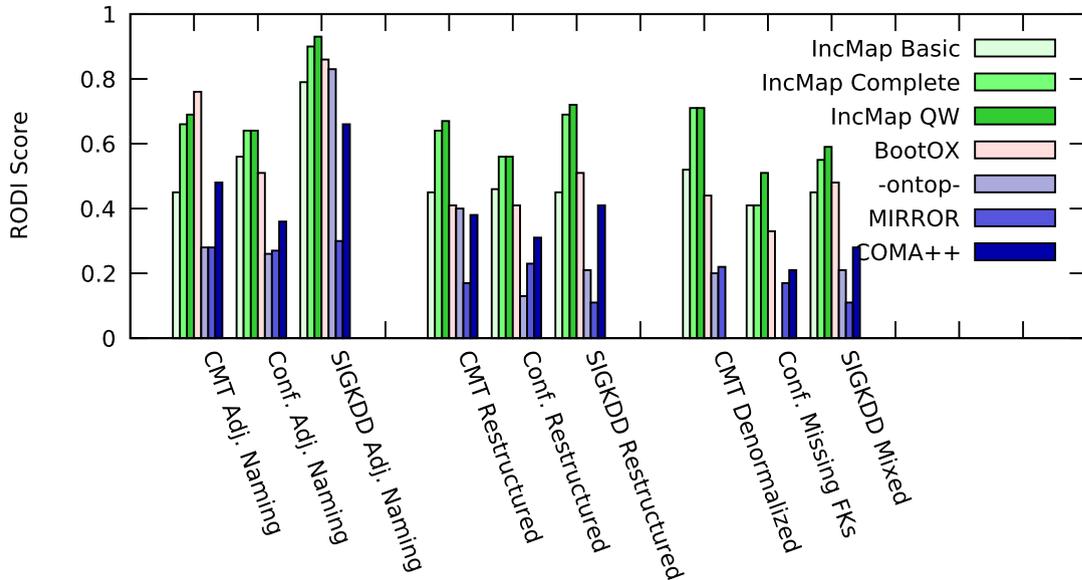


FIGURE 5.4: Overview of result scores in default conference scenarios by different systems. Different  $i^3$ MAGE configurations in shades of green.

TABLE 5.5: Overall scores in conference adjusted naming scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	CMT	Conf.	SIGKDD
IncMap Basic	0.45	0.56	0.79
IncMap Complete	0.66	<b>0.64</b>	0.90
IncMap QW	0.69	<b>0.64</b>	<b>0.93</b>
BootOX	<b>0.76</b>	0.51	0.86
-ontop-	0.28	0.26	0.38
MIRROR	0.28	0.27	0.30
COMA++	0.48	0.36	0.66

trade-offs need to be considered between the different versions of IncMap: without exception, more advanced versions with additional core features score at least as well as any of the less advanced versions, for all RODI benchmark scenarios.

As another observation, however, we notice that  $i^3$ MAGE, even while outperforming other systems, significantly struggles with most of the more complex scenarios. This matches another general impression on all tested systems, namely, that systems manage to solve some parts of the mapping scenarios, but with declining success as scenario complexity increases.

Figure 5.4 gives an overview of results in main conference domain scenarios, with complexity of scenario types increasing from left (adjusted naming) to right (special challenges).

TABLE 5.6: Overall scores in conference restructured scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	CMT	Conf.	SIGKDD
IncMap Basic	0.45	0.46	0.45
IncMap Complete	0.64	<b>0.56</b>	0.69
IncMap QW	<b>0.67</b>	<b>0.56</b>	<b>0.72</b>
BootOX	0.41	0.41	0.52
-ontop-	0.14	0.13	0.21
MIRROR	0.17	0.23	0.11
COMA++	0.38	0.31	0.41

TABLE 5.7: Overall scores in conference special case scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	CMT	Conf.	SIGKDD
	Denormalized	Missing FKs	Combined
IncMap Basic	0.52	0.41	0.45
IncMap Complete	<b>0.71</b>	0.41	0.55
IncMap QW	<b>0.71</b>	<b>0.51</b>	<b>0.59</b>
BootOX	0.44	0.33	0.48
-ontop-	0.20	-	0.21
MIRROR	0.22	0.17	0.11
COMA++	-	0.21	0.28

This overall picture also shows in individual numbers. For instance, relational schemata in the conference adjusted naming scenarios follow modeling patterns from their corresponding ontologies very closely (Table 5.5). Consequently, all systems without exception generally perform best in this part of the experiments. Quality drops for other types of scenarios, i.e., whenever we introduce additional challenges that are specific to the RDB2RDF modeling gap. The drop in accuracy between adjusted naming and restructured scenarios (Table 5.6) is to the most part due to the  $n:1$  mapping challenge. This challenge is introduced by one of the relational patterns that represent class hierarchies, namely, the one which groups data for several subclasses in a single table. In even more advanced conference cases (Table 5.7), systems tend to lose further due to the additional challenges, although to different degrees.

Table 5.8 showcases results from the semantically most heterogeneous scenarios in the conference domain. All of them are built on the “combined case” scenarios, i.e., they contain a mix of all of the standard RDB2RDF mapping challenges except for denormalization and lazy modeling of constraints. In addition, they increase the level of semantic heterogeneity by asking for mappings between a schema derived from one ontology to a completely different and independent ontology in the same domain.

TABLE 5.8: Overall scores in cross-matching scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	Conf. to CMT	SIGKDD to CMT	CMT to Conf.	SIGKDD to Conf.	CMT to SIGKDD	Conf. to SIGKDD
IncMap Basic	0.35	0.33	0.34	0.30	0.51	0.44
IncMap Complete	0.40	0.52	<b>0.69</b>	<b>0.42</b>	0.64	<b>0.65</b>
IncMap QW	<b>0.45</b>	<b>0.57</b>	<b>0.69</b>	<b>0.42</b>	<b>0.69</b>	<b>0.65</b>
BootOX	0.20	0.33	0.20	0.13	0.46	0.22
-ontop-	0.10	0.19	0.05	0.09	0.19	0.13
MIRROR	0.00	0.00	0.00	0.00	0.00	0.00
COMA++	0.00	0.14	0.05	0.04	0.24	0.09

Scores achieved by all systems are generally lower than in the basic conference cases discussed above. Reasonable scores can still be achieved by some tested systems. The most notable and surprising observation is that  $i^3$ MAGE outperforms all other systems in all of its configurations. That is, even  $i^3$ MAGE running IncMap Basic achieves better scores than any non- $i^3$ MAGE competitor in any of the scenarios that focus on semantic heterogeneity.

Why this is the case remains partially unclear to us, even after inspecting individual per-query results for several systems. In fact, the scores achieved by  $i^3$ MAGE setups are broadly in line with expectations, i.e., they are generally somewhat lower than in the comparable basic scenarios tested above, but not massively. The same largely holds for -ontop-. For BootOX, the relatively low scores can be partially explained by the fact that those scenarios combine all challenges from both “adjusted naming” and “restructured” scenarios. The weakness that BootOX exhibits in “restructured” scenarios also takes its toll on the overall scores in cross-matching scenarios. More surprisingly, COMA++ also loses out more than other contenders. COMA++ works in a similar manner to IncMap Basic in many ways, i.e., it matches directly between the relational and ontology models and uses model-independent graph matching to achieve this goal. Although  $i^3$ MAGE optimizes its matching graph for RDB2RDF, where COMA++ uses general-purpose graphs, the most relevant optimizations are only introduced in IncMap *Complete*, and we had therefore expected to see a relatively similar behavior between IncMap Basic and COMA++. We inspected the match output produced by COMA++ in search for an explanation and could rule out an issue introduced by our own translation of matches into R2RML mappings. It was not possible for us to debug and understand the matching process inside COMA++. Possibly, some other settings of COMA++ could have led to a more consistent behavior, but we were using a configuration that has been confirmed by the authors as suitable under the given circumstances. Also, a few other variations

TABLE 5.9: Overall scores in geodata scenario (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	Geodata
IncMap Basic	0.08
IncMap Complete	0.08
IncMap QW	<b>0.28</b>
BootOX	0.13
-ontop-	-
MIRROR	-
COMA++	-

TABLE 5.10: Overall scores in oil &amp; gas scenarios (scores based on average of per-test F-measure). Best numbers per scenario in bold print.

System	Oil & Gas User	Oil & Gas Atomic
IncMap Basic	0.00	0.12
IncMap Complete	0.00	<b>0.17</b>
IncMap QW	<b>0.06</b>	<b>0.17</b>
BootOX	0.00	0.14
-ontop-	0.00	0.10
MIRROR	0.00	0.00
COMA++	0.00	0.02

that we tried did not give better overall results. For MIRROR we also cannot offer any plausible explanation for the sharp drop compared to the basic conference scenarios above (no tasks solved at all). This drop is unexpected in part because MIRROR essentially applies the same ontology alignment technology as BootOX for matching, but performs even worse on these tasks. It could be an indicator that out-of-the-box ontology alignment techniques could not take the same leverage that they do when aligning original ontologies. We suspect that some different setup of ontology alignment exists, which would be more favorable for using the intermediate ontologies produced by MIRROR.

### 5.6.3.2 Overall Results for Geodata and Oil & Gas

While all of the conference scenarios test a wide range of specific RDB2RDF mapping challenges, they do so in a highly controlled fashion. Schemata are of medium size and complexity, and the query workload used is largely simplified. For instance, queries in the conference domain scenarios would separately check for mappings of authors, person names, and papers. They would not, however, pose any queries like asking for the names of authors who did participate in at least five different papers. The huge difference here is that, if two out of three of these elements were mapped correctly, the simple atomic

queries would report an average score of 0.66, while the single, more application-like query that correlates the same elements would not retrieve anything, thus resulting in a score of 0.00. This kind of real-world queries that mimic an actual application query workload is the focus of the remaining RODI default scenarios, which are set in the geodata and oil & gas exploration domains, respectively. Consequently, scores are lower again in those scenarios.

In the geodata scenario (Table 5.9), only a minority of query tests could be solved. Detailed debugging did show the reason for this to be in the nature of queries, most of which go beyond returning simple results of just a single mapped element. Another related reason is the use of some generic datatype properties such as names, with either fairly generic domains or complex unions as their domain. The kind of  $n:1$  mapping required for them can easily be mismatched, yet those properties are extensively used in many of the queries. This is also the reason, why in this scenario i<sup>3</sup>MAGE with IncMap QW performs significantly better than with any other version of IncMap, but also clearly outperforms all other tested systems. Among the two advantages that IncMap gains by accessing the query workload, limiting the target ontology to a relevant core and disambiguating relevant domains and ranges, the second proved to be more important in this scenario, as we found out by inspecting individual per-query results and internal match processing. Note, that -ontop-, MIRROR and COMA++ failed to load or process the target ontology and therefore did not produce any results.

In the oil & gas case (Table 5.10), the situation becomes even more difficult than for geodata. Here, the schema and ontology are again a bit more complex than in the geodata scenario, and so is the explorative query workload (“user queries”). None of the systems was able to answer any of these queries correctly after a round of automatic mapping, except for IncMap QW, which managed to solve one out of 17 query tests. After inspecting individual per-query results and internal match processing we suspect that IncMap QW could have the potential for solving an even greater number of tests in similar situations, but a degree of effective randomness involved in choosing between similar matches for mapping generation did not play out in its favor on several occasions for this scenario. In contrast to the geodata scenario, the positive effect of knowing the query workload in this case was chiefly a result of limiting the (rather large) target ontology to a relevant core.

To retrieve more meaningful results on oil & gas data, we added a second scenario on the same data, but with a synthetic query workload of atomic queries (“atomic”), which covers most of the schema and ontology. On this scenario, results could be computed. Overall scores remain however low. This is mostly due to the size and complexity of the schema and ontology with a large search space, hinting at the general need to

improve matching and mapping generation especially for large and complex schemata. In addition, the scenario requires a high number of 1: $n$  matches, testing this challenge much more thoroughly than any other scenario in the benchmark.

### 5.6.3.3 Drill-down on Selected Challenges

TABLE 5.11: Score break-down for queries on different match types with adjusted naming conference scenarios. 'C' stands for queries on classes, 'D' for data properties, 'O' for object properties.

System	CMT			Conference			SIGKDD		
	C	D	O	C	D	O	C	D	O
IncMap Basic	0.58	0.46	0.17	<b>0.81</b>	0.53	0.13	<b>1.00</b>	0.70	0.25
IncMap Complete	0.75	0.73	0.33	<b>0.81</b>	<b>0.67</b>	0.25	<b>1.00</b>	0.80	<b>0.75</b>
IncMap QW	0.75	<b>0.82</b>	0.33	<b>0.81</b>	<b>0.67</b>	0.25	<b>1.00</b>	<b>0.90</b>	<b>0.75</b>
BootOX	<b>0.92</b>	0.73	<b>0.50</b>	<b>0.81</b>	0.27	<b>0.38</b>	<b>1.00</b>	<b>0.90</b>	0.25
-ontop-	0.67	0.00	0.00	0.63	0.00	0.00	0.73	0.00	0.00
MIRROR	0.56	0.00	0.00	0.53	0.00	0.00	0.46	0.00	0.00
COMA++	0.75	0.46	0.00	0.50	0.40	0.00	0.80	0.70	0.00

All systems perform better for identifying class correspondences than they do for correctly identifying properties, as Table 5.11 shows. A further drill-down shows that this is in part due to the challenge of normalization artifacts, with systems struggling to detect properties that map to multi-hop join paths in the tables. Mapping data to class types appears to be generally easier for all contenders. i<sup>3</sup>MAGE and BootOX are performing best in most cases with all kinds of properties.

For i<sup>3</sup>MAGE, this drill-down highlights in particular where and how the effects of custom reasoning rules and advanced patterns brought by IncMap Complete affect the mapping: they improve the quality of property mappings, and object property mappings in particular. While the score gains from reasoning rules and patterns for all three categories, the gain for object properties is much more significant with at least about two times more correctly solved tests. Manual inspection of per-query results and debugging of the internal match generation process additionally indicates that some of the gains for class matches indirectly also result from better object property matches, as they support correct matches for their respective domain and range classes during structural matching.

The drill-down also gives some indication about the impact of a known query workload in these scenarios with IncMap QW: in some cases they improve the score for datatype properties. Manual inspection confirms that this is due to the correct disambiguation of relevant concrete domains for these properties, e.g., where *rdfs:labels* are used as a

TABLE 5.12: Score break-down for queries that test  $n:1$  matches in restructured conference domain scenarios.  $1:1$  and  $n:1$  stand for queries involving  $1:1$  or  $n:1$  mappings among classes and tables, respectively.

System	CMT		Conference		SIGKDD	
	1:1	n:1	1:1	n:1	1:1	n:1
IncMap Basic	0.79	0.00	<b>0.89</b>	0.00	0.86	0.00
IncMap Complete	<b>1.00</b>	<b>0.60</b>	<b>0.89</b>	<b>0.14</b>	<b>1.00</b>	<b>0.38</b>
IncMap QW	<b>1.00</b>	<b>0.60</b>	<b>0.89</b>	<b>0.14</b>	<b>1.00</b>	<b>0.38</b>
BootOX	0.86	0.00	0.78	0.00	<b>1.00</b>	0.00
-ontop-	0.57	0.00	0.56	0.00	0.86	0.00
MIRROR	0.00	0.00	0.00	0.00	0.00	0.00
COMA++	0.58	0.00	0.56	0.00	0.86	0.00

TABLE 5.13: Score break-down for queries that require  $1:n$  class matches on the Oil & Gas atomic tests scenario.

System	Oil & Gas Atomic		
	1:1	1:2	1:3
IncMap Basic	0.20	0.01	0.03
IncMap Complete	<b>0.28</b>	<b>0.11</b>	<b>0.09</b>
IncMap QW	<b>0.28</b>	<b>0.11</b>	<b>0.09</b>
BootOX	0.17	<b>0.11</b>	0.07
-ontop-	0.10	0.09	0.07
MIRROR	0.00	0.00	0.00
COMA++	0.03	0.00	0.00

naming property. The same effect did not manifest for object properties in any of the conference domain scenarios, because they are using sufficiently specific domains and ranges in all of the ontologies in question. Also, the second possible positive effect of using a query workload, namely, to limit matching to a relevant core subset of the ontology, does not appear to have any visible effect. This makes sense given the generally medium sizes of ontologies in these scenarios and the relatively large subset covered by some of the queries.

Tables 5.12 and 5.13 show the behavior of systems for finding  $n:1$  and  $1:n$  matches between ontology classes and table content, respectively. We highlight the  $n:1$  case on restructured conference scenarios and  $1:n$  matches on the oil & gas scenario as they include the highest number of tests in the respective categories.

In both cases results for anything but the  $1:1$  case are low, with all systems failing the large majority of tests.

For IncMap Basic as well as for all non-i<sup>3</sup>MAGE systems,  $1:n$  matches the situation is slightly better than it is with  $n:1$  matches. However, IncMap Complete shows the

TABLE 5.14: Impact of incremental mappings: Numbers for IncMap Complete @k human interactions. Bold print numbers mark first reaching the peak.

Scenario	@0	@6	@12	@24
	<b>Conference adjusted naming</b>			
CMT	0.66	0.78	<b>0.96</b>	0.96
CONFERENCE	0.64	0.80	<b>0.83</b>	0.83
SIGKDD	0.90	0.93	<b>1.00</b>	1.00
	<b>Conference restructured</b>			
CMT	0.64	0.70	<b>0.78</b>	0.78
CONFERENCE	0.56	0.62	0.70	<b>0.74</b>
SIGKDD	0.69	0.69	<b>0.75</b>	0.75

opposite behavior. This is a significant result, because  $1:n$  matches can be composed in mapping rules by simply adding up several correct  $1:1$  matches. A correct mapping of  $n:1$  matches between classes and tables, on the other side, usually requires the much more challenging task of *filtering* from the table that holds entities of different types. To identify tables that use such a pattern is one of the specific features in IncMap Complete. For i<sup>3</sup>MAGE, the same argument could be technically made for  $1:n$  matches, and the score gain between IncMap Basic and IncMap Complete is also high. But while i<sup>3</sup>MAGE uses patterns to also identify  $1:n$  matches, other systems (BootOX and -ontop-) handle them *without* them, but with a similar degree of success, probably by adding up several individual  $1:1$  matches as described above. At the same time this is not the case for  $n:1$  matches, which we consider very hard to solve without patterns and which none of the tested systems other than i<sup>3</sup>MAGE succeed with.

#### 5.6.4 Incremental Experiments: Results

We have also conducted a series of incremental, semi-automatic RODI experiments with i<sup>3</sup>MAGE. For this line of experiments we use i<sup>3</sup>MAGE with IncMap Complete.

Table 5.14 shows results for an incremental run of i<sup>3</sup>MAGE, i.e., using an oracle to simulate human feedback in several iterations.

The number “@k” in table headers denotes the number of simulated human interactions that we performed before the overall mapping quality did reach the specified score. Consequently, “@0” describes the baseline with no human interaction, and scores coincide with those of i<sup>3</sup>MAGE in the previous experiments.

Numbers “@6” and “@12” increase constantly, demonstrating the ability of i<sup>3</sup>MAGE to successfully consume simple feedback and improve mappings with a relatively small number of iterations.

At the same time, no further improvements could be made with more iterations ( $> 12$ ) in all but one case. Manual inspection did show two reasons for this upper limit: (1) some more improvement could have been reached with a more exhaustive feedback strategy, involving repeated lists of suggestions on the same query test or longer initial lists of suggestions. However, such an approach would inevitably also involve higher human effort. And (2), to some part the upper limit demonstrates a limitation of  $i^3$ MAGE's mapping suggestions.  $i^3$ MAGE is incapable of suggesting certain mappings requested by the benchmark, because most complex mappings (1:n, or involving n-way joins) will only be constructed during matching, when heuristic patterns support them with a sufficiently high score.

### 5.6.5 Summary of Results

The big picture in fully automatic mapping generation on RODI default scenarios shows that  $i^3$ MAGE (with its different IncMap setups) is generally clearly leading the field.

Looking at the broader landscape of other tested systems, both of the two most specialized and actively developed tools,  $i^3$ MAGE and BootOX, can be considered the top contenders. Among those two, BootOX performs particularly well in scenarios where the inter-model gap between relational schema and ontology is relatively small (e.g., “adjusted naming”).  $i^3$ MAGE is gaining ground when more specific inter-model mapping challenges are added. Other systems achieve generally weaker results. For MIRROR and -ontop- it has to be noted, though, that these systems have been originally designed and optimized for a somewhat different task than the full end-to-end mapping generation setup tested with RODI. COMA++ keeps up well, given that it is no longer actively developed and improved. However, results clearly show a significant advantage of  $i^3$ MAGE over all of these systems, including COMA++. We can assume that this is, at least in part, because COMA++ has been constructed to support inter-model matching in general but has not been explicitly optimized for the specific case of RDB2RDF matching.

Among the different setups of IncMap used in  $i^3$ MAGE, a constant increase of mapping quality can be observed while features are added. In particular, IncMap Complete always performs at least as good as IncMap Basic, and IncMap QW always achieves at least the same score as IncMap Complete. In many cases there is a significant gain between IncMap Basic and IncMap Complete. This means, that the effect of custom reasoning rules and relational patterns is generally important. The additional increase in scores achieved by IncMap QW is modest in most cases, but significant gains can be observed in a few scenarios. In particular, query workload analysis with IncMap QW can increase

the score for large and complex schemata where only a small subset is relevant, and also for cases where the ontology uses properties with complex or poorly specified domains and ranges.

When evaluated semi-automatically, numbers show that i<sup>3</sup>MAGE is well suited to also being used in an interactive scenario with feedback. It can help users to improve mapping quality with little effort beyond the fully automatic baseline, up to a point where the assistance of a technical expert is required.

## Chapter 6

# i<sup>3</sup>MAGE Applications

In this chapter, we introduce i<sup>3</sup>MAGE use cases and applications. We present i<sup>3</sup>MAGE as a mapping generator that can be applied in complete end-to-end use cases and can therefore be built into different applications.

First, in Section 6.1 we give an overview of use cases and applications that involve i<sup>3</sup>MAGE. Next, Section 6.2 presents the primary application of i<sup>3</sup>MAGE, its use for suggestions in an ETL workflow with a mapping editor. Finally, we also introduce a secondary application that can make use of i<sup>3</sup>MAGE’s query-driven mode of interaction in Section 6.3.

### 6.1 Overview of i<sup>3</sup>MAGE Use Cases and Applications

i<sup>3</sup>MAGE has been motivated by RDB2RDF data integration use cases with complex schemata and ontologies in mind. It has been implemented and tested with different applications. Each of them makes use of some of the different modes of operation offered by i<sup>3</sup>MAGE.

#### 6.1.1 Use Cases

The Optique project<sup>1</sup> is an EU-sponsored research project with the aim of developing a platform for scalable end-user access to big data, with a focus on schema complexity. As a platform [142, 149], Optique includes active mapping management [150], which can provide mapping suggestions from i<sup>3</sup>MAGE.

---

<sup>1</sup><http://optique-project.eu>

Optique has been deployed and tested mainly in two industry use cases in the energy sector: at oil and gas company Statoil [8, 138] to access oil field exploration data, and at Siemens [139] for analyzing (partially streaming) data from gas turbines. *i<sup>3</sup>MAGE* has been installed in conjunction with demonstrations of these use cases.

Additional use cases have been elicited from standard data integration applications of Information Workbench [151], a semantic platform, which is also used as the technological foundation of Optique. Information Workbench contains specific modules for data integration and is used in different enterprise scenarios. Those data integration scenarios are typically of lower complexity compared to Optique use cases (e.g., meta data and business data in data center management). Nevertheless, they are adding to the scope of use cases in which we have tested *i<sup>3</sup>MAGE* in practice.

### 6.1.2 Application Environments

On the one hand, *i<sup>3</sup>MAGE* produces small-granular mapping suggestions, generated by IncMap, its core matching and mapping component. On the other hand, *i<sup>3</sup>MAGE* can also generate complete mappings in a fully automatic manner. Both capabilities are being used in different applications.

Suggestions can be used within a mapping editor that is a part of the DataOps data integration solution (Section 6.2). In the same application, fully automatic mappings can alternatively be used to bootstrap a collection of mapping rules prior to manual editing. The environment of DataOps therefore constitutes a conventional setup for both automatic and semi-automatic use of mappings generated by *i<sup>3</sup>MAGE*. That is, it follows a normally manual process for assembling mappings, which can be extended and varied with support of automatic and semi-automatic mechanisms.

In the more complex and versatile data integration environment of the Optique platform, *i<sup>3</sup>MAGE* suggestions can additionally be used in a query-driven mode of operation. That is, *i<sup>3</sup>MAGE* suggestions can be used to curate mappings ad-hoc when needed by a query (Section 6.3). In this case, traditional suggestions in a mapping editor are also possible while fully automatic mappings are not supported through *i<sup>3</sup>MAGE*.

## 6.2 *i<sup>3</sup>MAGE* Support in DataOps Mapping Editor

When making suggestions while mappings are being edited, *i<sup>3</sup>MAGE* is employed in the mapping editor [83, 131] of the DataOps ETL solution [82].

### 6.2.1 DataOps Overview

Individual components for OBDI are commonly available, but end-to-end solutions are generally rare. DataOps has been designed with the aim of being a commercial-grade “Anything-to-RDF semantic data integration toolkit” [82]. It is thus positioned as an OBDI solution. DataOps has been developed at fluid Operations<sup>2</sup> as part of the company’s strategic data integration solutions and builds on the technology of a semantic platform, Information Workbench [151]. It has already been used with dozens of Information Workbench systems and customer projects, where it has also repeatedly been deployed for productive use.

Traditionally, for analytical applications on large or complex information, data warehouse systems are used as a backend and data is loaded by ETL-style processes. Those systems share an important property for enterprise use. Packaged as readily deployable solutions, they include everything needed for end-to-end enterprise information integration: from assisted setup, over a broad selection of data access methods, graphical configuration interfaces, ETL pipeline editing, explorative debugging, execution and re-publishing, to comprehensive documentation and reliable support.

However, these traditional solutions also come with a significant downside: in classical data warehouses, a dedicated global warehousing schema needs to be designed, mappings need to be constructed, and the resulting schema must be documented and communicated to consumers. This comes at the price of either a limited set of accessible data and query support with little flexibility. Or, if the scope of access and flexibility need to increase, it requires even more effort for programming all the tasks and queries to be supported. Worse, with a number of data sources that quickly change in structure, maintenance for the resulting schema, mapping and queries can quickly become a nightmare.

OBDI, with its flexible graph model and vocabularies, is one possible and natural way to address this predicament.

It is the aim of the DataOps toolkit to leverage the particular potential of OBDI over other data integration approaches. To do so, it provides a data integration environment with comprehensive functionality and a focus on end-to-end usability, much the same as traditional enterprise data warehousing solutions do. Although certain OBDI components are commonly available, complete end-to-end solutions that cater to the requirements of a production environment are few. Some existing and established frameworks such as the Linked Data Integration Framework [152] focus on Web-scale data rather than in-house enterprise data. More recent examples, e.g., UnifiedViews [153], provide

---

<sup>2</sup><http://www.fluidops.com/>

integrated, easy to use interfaces but focus on processing of pre-existing RDF data. Many systems composed of loosely coupled special-purpose components (where setup or maintenance involve intense human effort) fail the one key requirement that motivates their use in the first place: to significantly reduce overall effort in configuration and maintenance.

DataOps attempts to deliver an end-to-end solution for ontology-based ETL data integration that supports seamless setup, configuration and maintenance procedures. It bundles important components for every step of the process as a single toolkit out of the box, fills any gaps between those components and offers an integrated user interface, built on industry-proven platform technology.

The toolkit supports the integration of both semantic and non-semantic data from a host of different formats, including relational databases, CSV, Excel, XML, JSON, existing RDF graphs and others. Additional source formats can be integrated through an extension mechanism. For instance, in the scope of a particular project, a specialized data source that allows to directly access results from the statistical software suite *R* has been developed. In addition, each data source can be accessed from different physical or logical locations. For instance, data may reside in local files, on network shares (which may additionally require authentication), on the Web accessible through HTTP, or even hidden behind custom protocols.

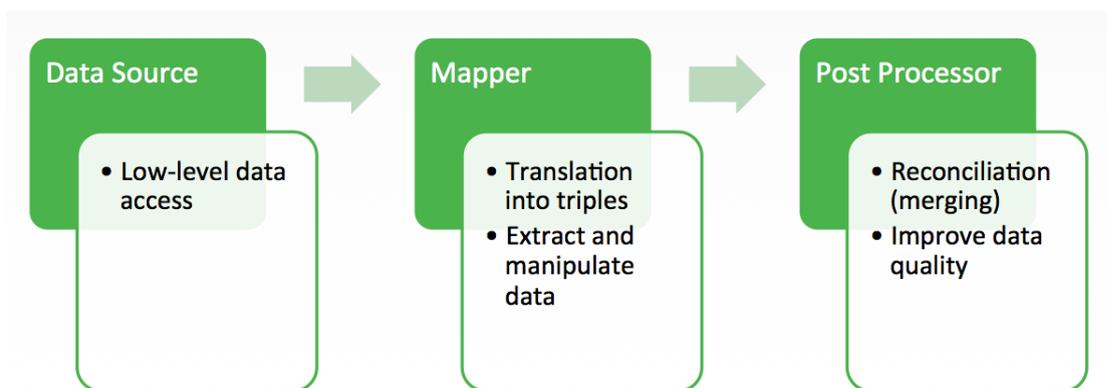


FIGURE 6.1: DataOps process

As an integrated toolkit, DataOps supports all setup, configuration and maintenance steps through a fully integrated Web interface with configuration forms and different mapping editors. Setting up data sources end-to-end is implemented in a three-step process (Figure 6.1):

1. Accessing the different data sources from arbitrary locations through different mechanisms (see screenshot in Figure 6.2a for details).

2. Specifying mappings depending on the data source format (Figure 6.2b). In the most common case of RDB2RDF, mappings are specified in R2RML using an integrated R2RML editor.
3. Consolidating new data with existing data instances, e.g., by establishing owl:sameAs links (Figure 6.2c).

For some of its features, DataOps makes use of established external components that are integrated in the backend. In particular, ETL extraction of relational databases currently relies on DB2Triples<sup>3</sup> and entity reconciliation uses Silk [154]. All modules are pluggable using generic interfaces and standards. Post-processor modules can even be stacked as pipelines of sub components. Other components are also in principle exchangeable. For example, other standard R2RML mapping engines can be hooked in, if required. The built-in interface used for editing RDB2RDF mappings itself is extensible in several ways. It builds on an initial prototype presented in [131], but has since been thoroughly remodeled [82].

### 6.2.2 *i<sup>3</sup>MAGE Application with DataOps*

DataOps supports plug-ins and extensions in different ways, e.g., through the Java service loader mechanism, method hook points, libraries, or UI customization. Specific features for data integration and business analytics use cases can then be bundled and installed as apps. Examples for this are more advanced visualization components, additional data sources, or, in the case of *i<sup>3</sup>MAGE*, automatic and semi-automatic mapping support.

The DataOps/Information Workbench *i<sup>3</sup>MAGE* app contains the complete JAR library of *i<sup>3</sup>MAGE* with all of its components. In addition, it contains platform integration components that are specific to the app:

- Suggestion translation using *i<sup>3</sup>MAGE*'s API, which materializes suggestions with the relevant meta data in an informal RDF vocabulary. Translation is triggered by *i<sup>3</sup>MAGE*'s API and RDF data is materialized through the platform data manager, which wraps around a Sesame OpenRDF API [137].
- Batch translation, pulling complete mappings from *i<sup>3</sup>MAGE*.
- A trigger method to invoke pulling automatic mappings and a UI mechanism to invoke this trigger. It is implemented by a platform component that is called a *Code*

---

<sup>3</sup><https://github.com/antidot/db2triples/>

Data Sources > Create a new Data Source.

The following form guides you through the configuration of a Data Source. Please follow the instructions on the right.

Data Source \*  ⓘ

Identifier \*  ⓘ

Location \*  ⓘ  
Choose File | no file selected

Xpath  ⓘ

Authentication ▶ show ⓘ

fields with a \* are required

(A) Step 1: configuration of data sources

Mapping Collection > Employees

### Edit Mapping Rule

Input data based on database table/view **PUBLIC.EMP** [ [change or extend](#) ]

1 - 1 / 10 | Show 1 rows (max. 1000)

UID	MBOX	FIRSTNAME	LASTNAME	DEPARTMENT	DOB	GENDER	PHONE	POSITION
1	pgonzales@acme.com	Pedro	Gonzales	Accounting	3/7/1985	M	0621495678348	Manager

**Subject**

**Subject is of Class**

**URI Template**  Edit  
`http://www.fluidops.com/resource/{UID}`

**You haven't configured any Predicate/Object maps.**  
Beside possible rdf:type triples for the instance subjects, no triples can be generated so far.

|

(B) Step 2: main mapping editor (R2RML)

Entity Reconciliation - Matching Rule Editor

↳ Display Example

Restrict existing entities to class: foaf:Person

Restrict incoming entities to class: foaf:Person

foaf:mbox equality :email

Confidence Threshold: average 1.0

Reconciliation Mode: LINK

(c) Step 3: reconciliation rules

FIGURE 6.2: Configuration/editing steps for DataOps mappings

*Execution Widget*. That is, a link is implemented through simple UI customizing and a click on the link uses a platform mechanism to trigger custom server-side Java code included with the *i<sup>3</sup>MAGE* app.

- An update trigger that gives implicit feedback based on manually created correspondences (selection of class types and/or tables in basic mapping rules) and triggers a fresh iteration of IncMap structural matching inside *i<sup>3</sup>MAGE*. The trigger uses method hookpoints.
- A dedicated UI component for visualizing *i<sup>3</sup>MAGE* suggestions (as materialized by the suggestion translation mechanism). This component is implemented through Information Workbench UI customization. It is thus deeply integrated with DataOps and its underlying platform.
- Suggestion confirmation and feedback handling, which is implemented alongside the UI component to visualize suggestions. Essentially, confirmation (or rejection) of suggestions and thus feedback to *i<sup>3</sup>MAGE* is triggered by a simple link that is again implemented by a platform Code Execution Widget.

### 6.2.2.1 Bootstrapping

There are two modes in which *i<sup>3</sup>MAGE* can be used in the DataOps app. With the first and more coarse-grained option, *i<sup>3</sup>MAGE* is used to *bootstrap* mappings.

For each data source, DataOps supports mapping creation either by creating an empty *mapping collection* (i.e., a set of R2RML mapping rules) or by bootstrapping a mapping collection. Bootstrapping a mapping collection by default means that DataOps will apply a variant of the W3C's direct mapping [109] to naively generate mapping rules for all data in the relational schema.

The *i<sup>3</sup>MAGE* app for DataOps offers an alternative bootstrapping method for DataOps mapping collections, which instead applies the mappings that *i<sup>3</sup>MAGE* generates fully automatically.

### 6.2.2.2 Mapping Suggestions

As a second option, mapping suggestions can make use of *i<sup>3</sup>MAGE*'s semi-automatic mode of operation. In this case, DataOps would simply create empty mapping collections and suggestions can appear while a user authors or modifies individual mapping rules manually.

Mapping Collection > Employees

### Edit Mapping Rule

Input data based on database table/view **PUBLIC.EMP** [ [change or extend](#) ]

UID	MBOX	FIRSTNAME	LASTNAME	DEPARTMENT	DOB	GENDER	PHONE	POSITION
1	pgonzales@acme.com	Pedro	Gonzales	Accounting	3/7/1985	M	0621495678348	Manager

**Suggested mappings:**

- **Class type :Employee, one per EMP.UID**
- Construct :Employee(s) from this table (EMP), use table attribute UID to make :Employee URIs unique.
- **Class type :Customer, one per EMP.UID**
- Construct :Customer(s) from this table (EMP), use table attribute UID to make :Customer URIs unique.
- **Class type foaf:Person, one per EMP.UID**
- Construct foaf:Person(s) from this table (EMP), use table attribute UID to make foaf:Person URIs unique.

**Subject**

**Subject is of Class**

**URI Template**

<http://www.fluidops.com/resource/{UID}>

You haven't configured any Predicate/Object maps.  
Beside possible rdf:type triples for the instance subjects, no triples can be generated so far.

[+ Add Predicate/Object Mapping](#) | [Preview Resulting Triples](#)

(A) Type suggestions: create instances of a specific type

Mapping Collection > Employees

### Edit Mapping Rule

Input data based on database table/view **PUBLIC.EMP** [ [change or extend](#) ]

UID	MBOX	FIRSTNAME	LASTNAME	DEPARTMENT	DOB	GENDER	PHONE	POSITION
1	pgonzales@acme.com	Pedro	Gonzales	Accounting	3/7/1985	M	0621495678348	Manager

**Subject**

**Subject is of Class**

foaf:Person

**URI Template**

<http://www.fluidops.com/resource/Person-{UID}>

**Suggested mappings:**

- **Add foaf:mbox from EMP.MBOX**
- For each foaf:Person add triples with foaf:mbox, using values from table attribute MBOX.
- **Add foaf:name from EMP.FIRSTNAME**
- For each foaf:Person add triples with foaf:name, using values from table attribute FIRSTNAME.
- **Add foaf:name from EMP.LASTNAME**
- For each foaf:Person add triples with foaf:name, using values from table attribute LASTNAME.

You haven't configured any Predicate/Object maps.  
Beside possible rdf:type triples for the instance subjects, no triples can be generated so far.

[+ Add Predicate/Object Mapping](#) | [Preview Resulting Triples](#)

(B) Predicate/object suggestions: create triples (properties and object values) for instances of a specific type

FIGURE 6.3: *i*<sup>3</sup>MAGE suggestions in DataOps

Figure 6.3 shows how such suggestions look in DataOps in practice. In the editing view of a freshly created or partially incomplete mapping rule, *i*<sup>3</sup>MAGE asks whether to complement the mapping according in some way. Suggestions are short-listed to display a top-3 of best ranked applicable suggestions. Each suggestion is labeled by a brief headline in bold-print, which should normally be sufficient to understand its purpose. A more verbose explanation of its effects is added in small print, mostly to reassure first-time users or occasional users. For each suggestion, two linked icons allow to accept and apply, or to reject and remove the suggestion. Accepted suggestions will immediately

modify and reload the mapping rule. Rejected suggestions are removed from the list and another suggestion, the one which is next in line according to its calculated score, will be displayed instead. In both cases, feedback is provided to *i<sup>3</sup>MAGE*.

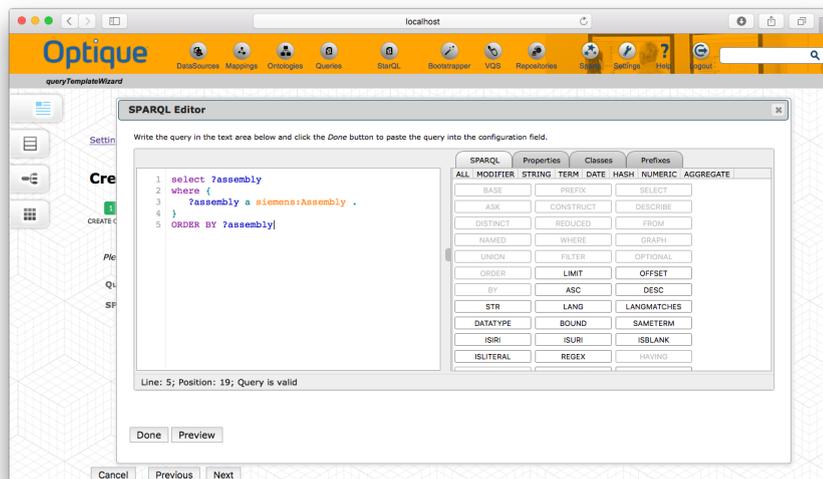
Due to the primary workflow of DataOps, suggestions are limited to appear in only two contexts: (1) to suggest IRI construction rules and ontology class types for R2RML mapping rules that have already a relational table selected (Figure 6.3a), and (2) for suggesting additional property/attribute matches in the context of existing R2RML mapping rules (Figure 6.3b).

While suggestions in these situations fit well with the manual mapping editing process of DataOps, they are not the only ones that *i<sup>3</sup>MAGE* could provide in general and they are not necessarily the most effective ones, either. In particular, a user always needs to manually identify a table from the database to map from, which can be difficult and tedious with complex schemata [131]. *i<sup>3</sup>MAGE* does identify matches between classes and tables and could thus in principle help with the selection. However, this would require a “class type first”, ontology-driven editing approach, which is not supported in the standard edition of DataOps. Experimentally, we have added such other kinds of suggestions, e.g., proposing complete mappings for a particular ontology class. They have not become a part of the app, though, as they have to be displayed in views that do belong to the core platform, not DataOps, and can therefore be confusing in other contexts.

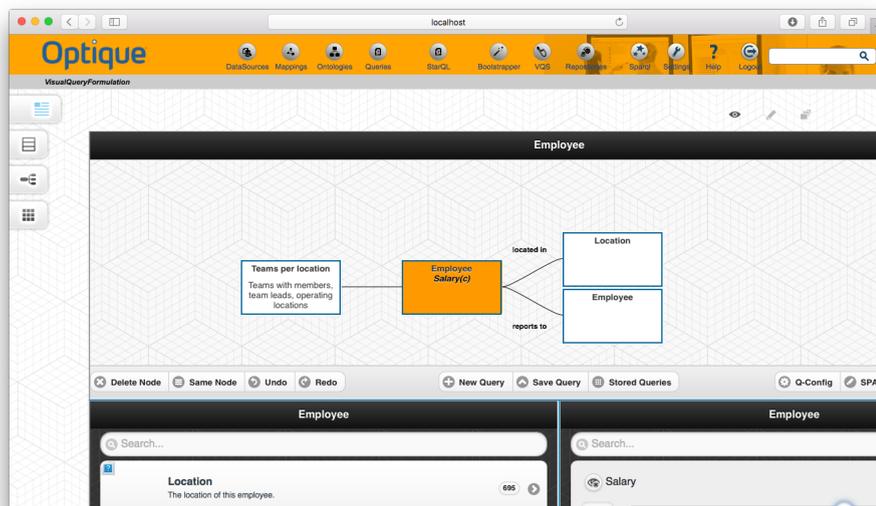
### 6.3 *i<sup>3</sup>MAGE* in Query Driven Setups

*i<sup>3</sup>MAGE* is also used for generating mappings within Optique ([138, 149]). Optique is a research prototype for scalable, semantic data integration and end-user access. Like DataOps, it builds on the platform technology of Information Workbench and integration therefore works in a similar way. Consequently, the same types of suggestions that *i<sup>3</sup>MAGE* makes for DataOps can also be made for Optique and have been demonstrated in [138]. As Optique brings its own bootstrapping capabilities [19], *i<sup>3</sup>MAGE* bootstrapping has not been deployed with Optique.

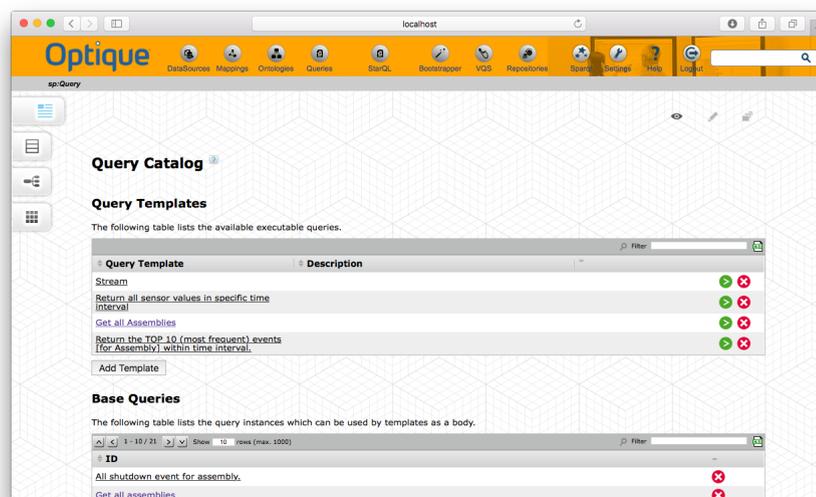
Optique forms a holistic data integration suite and offers a number of advanced components and more versatile workflows, which inspire further applications of *i<sup>3</sup>MAGE*. In particular, query formulation for analysis and customizing is supported through both an administrative and an end-user UI component in Optique [155].



(A) Administrative editing of queries in Optique



(B) Optique visual query formulation editor



(C) Information Workbench query catalog, running in Optique

FIGURE 6.4: Authoring queries in Optique

Figure 6.4 shows different options to author or edit queries in Optique. Queries can be authored in the usual expert way as plain SPARQL queries<sup>4</sup> (Figure 6.4a) or they can be edited visually even by non-experts (Figure 6.4b). Eventually, queries are stored in a platform query catalog (Figure 6.4c), where they can be directly executed or called upon for UI customization.

In particular the visual editing approach motivates the use of *i<sup>3</sup>MAGE*'s query-driven mode: if a mapping is required for one particular query only, *i<sup>3</sup>MAGE* can analyze the query and provide targeted suggestions as discussed previously in Section 4.2.12. Visual query editing is a particularly good fit for this approach for two reasons. Firstly, visual query formulation targets domain experts who are power-users but do not possess the knowledge and skills to write queries, administer a database, or author mappings. They are therefore specifically dependent on support, either automated or by technical experts, whenever they run in a situation where mappings are incomplete. And secondly, the limitations of query parsing in *i<sup>3</sup>MAGE* as discussed in Section 4.6 play no role in practice with visual query formulation, as the editor produces straight-forward queries of limited complexity that contain no critical information outside the scope of what *i<sup>3</sup>MAGE* can parse.

With such a setup, mapping suggestions are employed to generate mappings ad hoc when a query is being phrased and makes use of unmapped ontology axioms. As soon as a query has been stored in the query catalog, *i<sup>3</sup>MAGE* can analyze it and suggest to map required concepts or properties.

The screenshot shown in Figure 6.5 demonstrates how queries look in the query catalog with this approach. If the query relies on any concept or property for which no existing mappings have been identified by the mapping analyzer, *i<sup>3</sup>MAGE* will provide suggestions to extend the mapping. As usual, these suggestions contain a brief summary as well as some explanation. They can be accepted or rejected with a single click.

Note, that suggestions provided in this application exceed the scope of suggestions from the DataOps app. Suggestions in Figure 6.5 propose a match of a table for a given concept instead of the other way around. Similarly, in some cases a suggestion might be to draw data from a SQL JOIN over *several* tables rather than from just one single table. Also, to map properties with a domain of a type that is already mapped, the suggestion would attempt to match *both* URI construction for the existing type of the triple subjects *and* value construction for the property at the same time. All of these suggestions immediately correspond to individual mapping rules generated by *i<sup>3</sup>MAGE*

---

<sup>4</sup>Technically, Optique also supports queries in a dedicated query language for semantic streams. These cannot be parsed by *i<sup>3</sup>MAGE*, though, and are thus not supported.

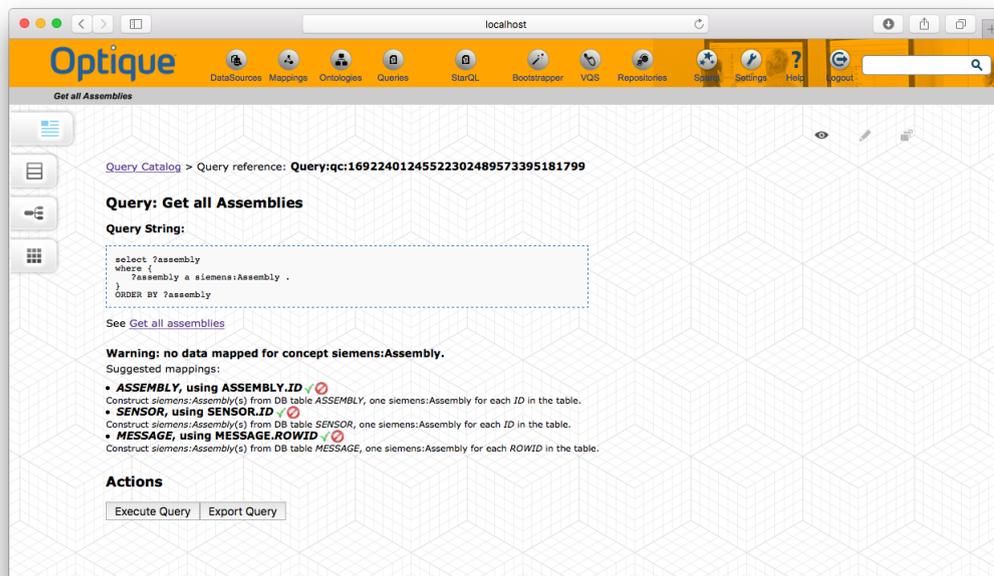


FIGURE 6.5: Query based mapping suggestions

from IncMap's internal mapping model. That is, they do not need to be explicitly composed or otherwise post-processed to suit the purpose.

In addition to the better use that this application makes of *i*<sup>3</sup>MAGE suggestions (compared with DataOps), it also benefits from the additional advantages of *i*<sup>3</sup>MAGE's query driven mode: queries steer the system to consider only areas of interest in the target ontology, and they may also help to disambiguate relevant domains and ranges of properties.

# Chapter 7

## Discussion

### 7.1 Summary

We have presented i<sup>3</sup>MAGE, a mapping generation system, which addresses the specific problems of incremental, interactive mapping generation in RDB2RDF inter-model settings. The system aims to reduce the overall human effort in the process of creating sufficiently accurate mappings. We generate mapping suggestions that are as close to the eventually expected mapping as possible, optimizing on particular technical challenges of the RDB2RDF inter-model use case. The system also attempts to provide suggestions in a way that makes them easy to process in user interactions and allows them to improve incrementally on user feedback. i<sup>3</sup>MAGE is built to fit into a wider mapping development process where other forms of support could be leveraged simultaneously.

We have designed and evaluated i<sup>3</sup>MAGE according to the following lead questions:

- *What are the specific challenges of inter-model mapping generation as opposed to generating regular intra-model mappings, specifically w.r.t. RDB2RDF mappings?*
- *How can mapping generation systems be designed to provide enhanced support for those specific RDB2RDF challenges?*
- *How can the quality of generated RDB2RDF mappings be measured w.r.t. real-world utility and how do specialized approaches compare to the state of the art?*
- *How can user-feedback and other context be exploited to gradually improve the quality of generated mappings?*
- *How can such generated mappings be integrated non-intrusively in a semi-automatic process?*

To this end, we have discussed specific challenges on the basis of a broader discussion about the technical background and related approaches that attempt to address those challenges (Chapter 2).

In the course of this discussion we have pointed out gaps and shortcomings that have not been sufficiently addressed to date. We have then proposed an approach that addresses those gaps and discussed its rationale (Chapter 3). We also introduced a novel system, i<sup>3</sup>MAGE, which implements our approach, and we have described this system and its technical foundations in detail (Chapter 4).

In order to evaluate our approach and system not only on grounds of feasibility but also in direct comparison to other approaches, we have analyzed existing benchmarks proposed in the literature and discussed the requirements for a broadly applicable end-to-end quality benchmark for RDB2RDF mapping generation. We did then design such a benchmark according to elicited requirements and provided a broad experimental evaluation of i<sup>3</sup>MAGE and several other systems with that benchmark (Chapter 5).

Additionally, we have analyzed potential opportunities for enhancing i<sup>3</sup>MAGE by making it incrementally semi-automatic and have considered additional information from interactive environments as matching context. We have proposed and implemented several such features along two related dimensions, incremental, pay-as-you-go development of mappings and interactive user feedback (c.f. Chapter 4). We have also separately evaluated the impact of those features (Chapter 5).

To demonstrate the applicability of i<sup>3</sup>MAGE in practice and to put the system into a wider context, we have analyzed potential uses of i<sup>3</sup>MAGE in real-world applications. We have presented a prototypical implementation of two such use cases and system environments and have discussed how i<sup>3</sup>MAGE is being used in these settings in practice (Chapter 6).

## 7.2 Discussion of Results

With our i<sup>3</sup>MAGE system, we have demonstrated the feasibility of a specialized inter-model RDB2RDF mapping generation system, which combines both generic structural commonalities between the different data models and specialized features to leverage specific properties of the particular models. The specific features are elicited from a thorough analytic discussion of both shared and distinct features of the different data models. No such approach has been proposed so far in the literature to the best of our knowledge.

In addition, we have analyzed potential opportunities of enhancing i<sup>3</sup>MAGE by making it incrementally semi-automatic and by considering additional information from interactive environments as matching context. While not the main research focus of this work, these additions are highly important to optimize the utility and usability of i<sup>3</sup>MAGE and can be seen as a precondition for its deployment in real-world application scenarios.

Based on our discussion and implementation of i<sup>3</sup>MAGE, we have gained experimental insights into the strengths and weaknesses of automatic and semi-automatic RDB $\mathcal{2}$ RDF mapping generation in general and into the performance of i<sup>3</sup>MAGE in particular. For RDB $\mathcal{2}$ RDF, no previous evaluations on generated mapping quality did compare a significant number of different systems and approaches. Among other findings, we could identify a number of particularly hard challenges that are difficult to solve for almost all systems. Besides a number of generally tough data integration challenges, several of those are specific features of the RDB $\mathcal{2}$ RDF inter-model gap, e.g., a type hierarchy pattern, where several types are modeled jointly in the same relational table. These findings support our hypothesis that special adjustments of matching algorithms for RDB $\mathcal{2}$ RDF may help in achieving better quality of generated mappings. Experiments could also provide evidence that interactive, incremental features such as user feedback could improve the results provided by i<sup>3</sup>MAGE.

In direct comparison to other approaches i<sup>3</sup>MAGE performs about as well as the best competing approaches, even in its basic version. When used with all advanced features enabled i<sup>3</sup>MAGE typically outperforms all other approaches.

However, we could also identify cases, where the performance of all known approaches, including i<sup>3</sup>MAGE, is still modest, and their usability in practice is questionable. This chiefly concerns cases of large and complex data sources with complex resulting mappings, i.e., situations where a large number of potential match candidates are difficult to disambiguate.

In addition to a principled experimental evaluation, we have demonstrated how i<sup>3</sup>MAGE could be employed in the context of holistic data integration systems and use cases. From these implementations we have learned that different modes of operation are required from i<sup>3</sup>MAGE to smoothly fit into different contexts. For instance, fully automatic mapping generation can only be useful in one specific type of mapping curation workflows. This also means that not all features of i<sup>3</sup>MAGE could be effectively employed in all application contexts. The modular and complementing nature of these features, however, makes i<sup>3</sup>MAGE usable in significantly different setups and configurations, and thus in several different types of applications.

### 7.3 Future Work

We have shown that i<sup>3</sup>MAGE can effectively generate RDB2RDF mappings and mapping suggestions and usually outperforms other approaches in terms of mapping quality. However, the approach could still be improved on several counts.

Future work includes support for auto-tuning strategies to set the various knobs and configuration options in i<sup>3</sup>MAGE. In particular, the best choice of lexical matching, graph edge weight factors and a number of Similarity Flooding control options could first be learned on a large corpus of RDB2RDF scenarios to provide improved default parameters. In incremental scenarios, parameter learning could continue during mapping creation to tailor parameters further to each individual case. The same applies to activation strategies for inactive edges and their respective activation thresholds.

Along the lines of the same idea, i.e., to use learning techniques, we could also likely improve the impact of relational patterns. We use complex relational patterns to support candidate correspondences with a number of ontology axioms. To identify such patterns, we currently use heuristic rules that check on one or more characteristics of elements or data in the relational schema, e.g., on the number and types of columns or the number of rows in a table. While these rules are currently hand-written, both binary thresholds and the weights for combining them into the overall pattern heuristic could be learned. In addition, it might be worth exploring to not only tune parameters of hand-written patterns with learning techniques but also to try feature extraction to identify additional usable patterns.

Also, while we have improved i<sup>3</sup>MAGE significantly by detecting advanced modeling patterns in relational schemata, we only associate them with individual corresponding axioms on the ontology side. This is sufficient in those cases that we did identify as obvious matching candidates. However, complex modeling patterns are common in ontologies as well and they could be used as correspondence candidates in a similar manner as relational patterns. It is reasonable to assume that such a widening of the scope of potential correspondences could help to improve matching for a number of edge cases.

Besides these possible optimizations, additional forms of interactions and human feedback might be worth exploring in another avenue of future work. While a positive impact of feedback and incremental mapping strategies have been observed, this aspect could be extended in various ways, and we expect to see further improvements when doing so. For instance, the system might ask leading questions to gather explicit feedback on the quality of mapping suggestions. Asking a user to provide samples of missing results

might be another way for enhanced interactivity, which promises an overall improvement in mapping quality.

Finally, while our current approach is based on schema matching and mapping, future work could extend into additional elements of data mapping. This way, automatic matching could not only become more precise (by validating match candidates on data instances). It could, in addition, become more expressive. For instance, it could propose fundamental data transformations such as the concatenation of a *first name* and *last name* into a *full name* field.

# Bibliography

- [1] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.
- [2] Anant Bhardwaj, Souvik Bhattacharjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya Parameswaran. DataHub: Collaborative Data Science & Dataset Version Management at Scale. In *CIDR*, 2015.
- [3] Xin L. Dong and Divesh Srivastava. *Big Data Integration*. Morgan Claypool, 2013.
- [4] Vinod K. Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *SOCC*, 2013.
- [5] Boris Evelson, Holger Kisker, Martha Bennett, and Sophia Christakis. Benchmark Your BI Environment. Technical report, Forrester Research, Inc., October 2013.
- [6] Ian Horrocks. What Are Ontologies Good For? In *Evolution of Semantic Systems*. Springer, 2013.
- [7] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking Data to Ontologies. *J. on Data Sem.*, 10(1), 2008.
- [8] Evgeny Kharlamov, Martin Giese, Ernesto Jiménez-Ruiz, Martin G. Skjæveland, Ahmet Soylu, Dmitriy Zheleznyakov, Timea Bagosia, Marco Consolea, Peter Haase, Ian Horrocks, Sarunas Marciuska, Christoph Pinkel, Mariano Rodriguez-Muro, Marco Ruzzi, Valerio Santarelli, Domenico F. Savo, Kunal Sengupta, Michael Schmidt, Evgenij Thorstensen, Johannes Trame, and Arild Waaler. Optique 1.0: Semantic Access to Big Data – The Case of Norwegian Petroleum Directorate’s FactPages. In *ISWC (Posters & Demos)*, 2013.
- [9] Diego Calvanese, Alessandro Mosca, José Remesal, Martín Rezk, and Guillem Rull. A ‘Historical Case’ of Ontology-Based Data Access. In *DH*, 2015.

- [10] Evgeny Kharlamov, Nina Solomakhina, Özgür L. Özçep, Dmitriy Zheleznyakov, Thomas Hubauer, Steffen Lamparter, Mikhail Roshchin, and Ahmet Soylu. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In *ISWC*, 2014.
- [11] Cristina Civili, Marco Console, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Lorenzo Lepore, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, Valerio Santarelli, and Domenico F. Savo. MASTRO STUDIO: Managing Ontology-Based Data Access applications. In *VLDB*, 2013.
- [12] Michael Bada, Robert Stevens, Carole A. Goble, Yolanda Gil, Michael Ashburner, Judith A. Blake, J. Michael Cherry, Midori A. Harris, and Suzanna Lewis. A Short Study on the Success of the Gene Ontology. *J. Web Sem.*, 1(2), 2004.
- [13] Fred Freitas and Stefan Schulz. Survey of Current Terminologies and Ontologies in Biology and Medicine. *RECIIS – Elect. J. Commun. Inf. Innov. Health*, 3(1), 2009.
- [14] Martin G. Skjæveland, Espen H. Lian, and Ian Horrocks. Publishing the Norwegian Petroleum Directorate’s FactPages as Semantic Web Data. In *ISWC*, 2013.
- [15] Philip A. Bernstein, Sergey Melnik, and John E. Churchill. Incremental Schema Matching. In *VLDB*, 2006.
- [16] Hong-Hai Do and Erhard Rahm. COMA – A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, 2002.
- [17] Mauricio A. Hernández, Renée J. Miller, and Laura M. Haas. Clio: A Semi-automatic Tool for Schema Mapping. *ACM SIGMOD Record*, 30(2), 2001.
- [18] Craig A. Knoblock, Pedro Szekely, Jose L. Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, and Mohsen Taheriyani. Semi-Automatically Mapping Structured Sources into the Semantic Web. In *ESWC*, 2012.
- [19] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical Mapping of RDBs to OWL 2. In *ISWC*, 2015.
- [20] Aibo Tian, Juan F. Sequeda, and Daniel P. Miranker. QODI: Query as Context in Automatic Data Integration. In *ISWC*, 2013.
- [21] Luciano F. de Medeiros, Freddy Priyatna, and Oscar Corcho. MIRROR: Automatic R2RML Mapping Generation from Relational Databases. In *ICWE*, 2015.

- [22] Juan F. Sequeda and Daniel Miranker. Ultrawrap Mapper: A Semi-Automatic Relational-Database-to-RDF (RDB2RDF) Mapping Tool. In *ISWC (Posters & Demos)*, 2015.
- [23] Petros Papapanagiotou, Polyxeni Katsiouli, Vassileios Tsetsos, and Christos Anagnostopoulos. Ronto: Relational to Ontology Schema Matching. *AIS SIGSEMIS BULLETIN*, 3(3-4), 2006.
- [24] David Aumüller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and Ontology Matching with COMA++. In *SIGMOD*, 2005.
- [25] Ontology Alignment Evaluation Initiative. <http://oaei.ontologymatching.org>, 2005.
- [26] Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas O. Kempf, Patrick Lambrix, Stefano Montanelli, Heiko Paulheim, Dominique Ritzke, Pavel Shvaiko, Alessandro Solimando, Cássia Trojahn, Ondrej Zamazal, and Bernardo Cuenca Grau. Results of the Ontology Alignment Evaluation Initiative 2014. In *OM*, 2014.
- [27] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Information Retrieval*, 13(6), 1970.
- [28] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems – The Complete Book*. Prentice Hall, 2nd edition, 2008.
- [29] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview, 2009. URL <http://www.w3.org/TR/2012/REC-owl2-overview-20121211/>.
- [30] Balder ten Cate and Phokion G. Kolaitis. Structural Characterizations of Schema-mapping Languages. *Commun. ACM*, 53(1), 2010.
- [31] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. On Directly Mapping Relational Databases to RDF and OWL. In *WWW*, 2012.
- [32] Terry Landers and Ronni L. Rosenberg. An Overview of MULTIBASE. In *International Symposium on Distributed Data Bases*. North-Holland Publishing Company, 1982.
- [33] Gio Wiederhold. Mediators in the Architecture of Future Computer Systems. *IEEE Computer*, 25(3), 1992.
- [34] Philip A. Bernstein and Laura M. Haas. Information Integration in the Enterprise. *Commun. ACM*, 51(9), 2008.

- 
- [35] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise Information Integration: Successes, Challenges and Controversies. In *SIGMOD*, 2005.
- [36] Ronald E. Giachetti. A Framework to Review the Information Integration of the Enterprise. *Int. J. of Production Research*, 42(6), 2004.
- [37] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4), 1986.
- [38] Aris M. Ouksel and Amit Sheth. Semantic Interoperability in Global Information Systems. *ACM SIGMOD Record*, 28(1), 1999.
- [39] AnHai Doan and Alon Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1), 2005.
- [40] Renée J. Miller, Laura M. Haas, and Mauricio Hernandez. Schema Mapping as Query Discovery. In *VLDB*, 2000.
- [41] Jeffrey D. Ullman. Information Integration Using Logical Views. In *ICDT*, 1997.
- [42] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, 2002.
- [43] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4), 2001.
- [44] Michelle Cheatham and Pascal Hitzler. String Similarity Metrics for Ontology Alignment. In *ISWC*, 2013.
- [45] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *ICDE*, 2002.
- [46] Sandra Castano and Valeria De Antonellis. Global Viewing of Heterogeneous Data Sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2), 2001.
- [47] Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-based Matching Approaches. *J. on Data Sem.*, 4(1), 2005.
- [48] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic Schema Matching, Ten Years Later. In *VLDB*, 2011.
- [49] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *VLDB*, 2001.

- [50] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic Integration of Semistructured and Structured Data Sources. *ACM SIGMOD Record*, 28(1), 1999.
- [51] Isabel F. Cruz, Flavio P. Antonelli, and Cosmin Stroe. AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies. In *VLDB*, 2009.
- [52] Christoph Pinkel, Carsten Binnig, Evgeny Kharlamov, and Peter Haase. IncMap: Pay-as-you-go Matching of Relational Schemata to OWL Ontologies. In *OM*, 2013.
- [53] Christoph Pinkel, Carsten Binnig, Evgeny Kharlamov, and Peter Haase. Pay as you go Matching of Relational Schemata to OWL Ontologies with IncMap. In *ISWC (Posters & Demos)*, 2013.
- [54] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm (Extended Technical Report). Technical report, Stanford, 2001.
- [55] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*, 2005.
- [56] Ronald Fagin, Laura M. Haas, Mauricio Hernandez, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*. Springer, 2009.
- [57] Natalya F. Noy and Mark A. Musen. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *Int. J. of Human-Computer Studies*, 59(6), 2003.
- [58] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. LogMap: Logic-Based and Scalable Ontology Matching. In *International Semantic Web Conference*, 2011.
- [59] Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. A Probabilistic-Logical Framework for Ontology Matching. In *AAAI*, 2010.
- [60] Monika Lanzemberger and Jennifer Sampson. Alviz – A Tool for Visual Ontology Alignment. In *IV*, 2006.
- [61] Patrick Lambrix and He Tan. SAMBO – A System for Aligning and Merging Biomedical Ontologies. *J. Web Sem.*, 4(3), 2006.
- [62] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8), 2009.

- [63] DuyHoa Ngo and Zohra Bellahsene. YAM++: A Multi-strategy Based Approach for Ontology Matching Task. In *EKAW*, 2012.
- [64] Holger Wache, Thomas Voegelé, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. Ontology-based Integration of Information – A Survey of Existing Approaches. In *IJCAI*, 2001.
- [65] Pavel Shvaiko and Jérôme Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1), 2013.
- [66] Wei Hu and Yuzhong Qu. Discovering Simple Mappings Between Relational Database Schemas and Ontologies. In *ISWC/ASWC*, 2007.
- [67] Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d’Aquin, and Enrico Motta. The Neon Ontology Engineering Toolkit. In *WWW*, 2008.
- [68] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Bootstrapping OWL 2 Ontologies and R2RML Mappings from Relational Databases. In *ISWC (Posters & Demos)*, 2015.
- [69] Thanh Tran, Haofen Wang, and Peter Haase. Hermes: Data Web Search on a Pay-as-you-go Integration Infrastructure. *J. Web Sem.*, 7(3), 2009.
- [70] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go User Feedback for Dataspace Systems. In *SIGMOD*, 2008.
- [71] Martin Hentschel, Laura Haas, and Renée J. Miller. Just-in-time Data Integration in Action. In *VLDB*, 2010.
- [72] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale Data Integration: You Can Only Afford to Pay As You Go. In *CIDR*, 2007.
- [73] Aditya Parameswaran and Neoklis Polyzotis. Answering Queries using Humans, Algorithms and Databases. In *CIDR*, 2011.
- [74] Fernando Wagner, Jose A.F. Macedo, and Bernadette Lóscio. An Incremental and User Feedback-based Ontology Matching Approach. In *iiWAS*, 2011.
- [75] Patrick Lambrix and Qiang Liu. Using Partial Reference Alignments to Align Ontologies. In *ESWC*, 2009.

- [76] Juan F. Sequeda, Rudy Depena, and Daniel P. Miranker. Ultrawrap: Using SQL Views for RDB2RDF. In *ISWC*, 2009.
- [77] Souripriya Das, Seema Sundara and Richard Cyganiak (Eds.). R2RML: RDB to RDF Mapping Language, 2012. URL <http://www.w3.org/TR/2012/REC-r2rml-20120927/>.
- [78] Christian Bizer and Andy Seaborne. D2RQ – Treating non-RDF Databases as Virtual RDF Graphs. In *ISWC*, 2004.
- [79] Mariano Rodríguez-Muro and Martín Rezk. Efficient SPARQL-to-SQL with R2RML Mappings. *J. Web Sem.*, 2015.
- [80] Percy E. Salas, Edgard Marx, Alexander Mera, and Karin K. Breitman. RDB2RDF Plugin: Relational Databases to RDF Plugin for Eclipse. In *TOPI*, 2011.
- [81] Luís E. Neto, Vânia M. Vidal, Marco A. Casanova, and José M. Monteiro. R2RML by Assertion: A Semi-automatic Tool for Generating Customised R2RML Mappings. In *ESWC (Satellite Events)*, 2013.
- [82] Christoph Pinkel, Andreas Schwarte, Johannes Trame, Andriy Nikolov, Ana Sasa Bastinos, and Tobias Zeuch. DataOps: Seamless End-to-end Anything-to-RDF Data Integration. In *ESWC (Posters & Demos)*, 2015.
- [83] Kunal Sengupta, Peter Haase, Michael Schmidt, and Pascal Hitzler. Editing R2RML Mappings Made Easy. In *ISWC (Posters & Demos)*, 2013.
- [84] Sean M. Falconer and Natalya F. Noy. Interactive Techniques to Support Ontology Matching. In *Schema Matching and Mapping*. Springer, 2011.
- [85] Heiner Stuckenschmidt, Jan Noessner, and Faraz Fallahi. A Study in User-centric Data Integration. In *ICEIS*, 2012.
- [86] Sean M. Falconer and Margaret-Anne Storey. A Cognitive Support Framework for Ontology Mapping. In *ISWC*, 2007.
- [87] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *CHI*, 2011.
- [88] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.
- [89] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *J. ACM*, 42(4), 1995.

- [90] Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. Bringing Relational Databases into the Semantic Web: A Survey. *Semantic Web*, 3(2), 2012.
- [91] Won Kim. *Introduction to Object-oriented Databases*. MIT press Cambridge, 1990.
- [92] Kathi H. Davis and Adarsh K. Arora. Converting a Relational Database Model into an Entity-Relationship Model. In *Int. Conf. on Entity-Relationship Approach*, 1987.
- [93] Roger H.L. Chiang, Terence M. Barron, and Veda C. Storey. Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database. *Data & Knowledge Engineering*, 12(2), 1994.
- [94] Hausi Müller, Jens Jahnke, Dennis Smith, Margaret-Anne Storey, Scott Tilley, and Kenny Wong. Reverse Engineering: A Roadmap. In *ICSE*, 2000.
- [95] Ankit Malpani, Philip A. Bernstein, Sergey Melnik, and James F. Terwilliger. Reverse Engineering Models from Databases to Bootstrap Application Development. In *ICDE*, 2010.
- [96] Irina Astrova. Reverse Engineering of Relational Databases to Ontologies. In *ESWC*, 2004.
- [97] Freddy Priyatna and Boris Villazón-Terrazas. Building Ontologies by Using Re-engineering Patterns and R2RML Mappings. In *WOP*, 2012.
- [98] Juan F. Sequeda, Freddy Priyatna, and Boris Villazon-Terrazas. Relational Database to RDF Mapping Patterns. In *WOP*, 2012.
- [99] Jun-Ki Min, Jae-Yong Ahn, and Chin-Wan Chung. Efficient Extraction of Schemas for XML Documents. *Information Processing Letters*, 85(1), 2003.
- [100] Christian Bizer. D2R MAP – A Database to RDF Mapping Language. In *WWW (Posters)*, 2003.
- [101] Jesús Barrasa Rodríguez, Oscar Corcho, and Asunción Gómez-Pérez. R<sub>2</sub>O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. In *SWDB*, 2004.
- [102] Matthias Hert, Gerald Reif, and Harald C. Gall. A Comparison of RDB-to-RDF Mapping Languages. In *SEMANTiCS*, 2011.
- [103] Claus Stadler, Jörg Unbehauen, Patrick Westphal, Mohamed Sherif, and Jens Lehmann. Simplified RDB2RDF Mapping. In *LDOW*, 2015.

- [104] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van De Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*, 2014.
- [105] Eduard C. Dragut and Ramon Lawrence. Composing Mappings Between Schemas Using a Reference Ontology. In *CoopIS/DOA/ODBASE*, 2004.
- [106] Nenad Stojanovic, Ljiljana Stojanovic, and Raphael Volz. A Reverse Engineering Approach for Migrating Data-intensive Web Sites to the Semantic Web. In *IFIP*, 2002.
- [107] Andreas Behm, Andreas Geppert, and Klaus R. Dittrich. On the Migration of Relational Schemas and Data to Object-oriented Database Systems. In *Int. Conf. on Re-Technologies in Information Systems*, 1997.
- [108] Yuan An, Alex Borgida, and John Mylopoulos. Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences. In *ODBASE*, 2005.
- [109] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux and Juan Sequeda (Eds.). A Direct Mapping of Relational Data to RDF, 2012. URL <http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/>.
- [110] Nadine Cullot, Raji Ghawi, and Kokou Yétongnon. DB2OWL: A Tool for Automatic Database-to-Ontology Mapping. In *SEBD*, 2007.
- [111] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn dos Santos. Ontology Alignment Evaluation Initiative: Six Years of Experience. *J. on Data Sem.*, 15(1), 2011.
- [112] Salvatore Raunich and Erhard Rahm. Towards a Benchmark for Ontology Merging. In *EI2N*, 2012.
- [113] Carlos R. Rivero, Andreas Schultz, and Christian Bizer. Benchmarking the Performance of Linked Data Translation Systems. In *LDOW*, 2012.
- [114] Bogdan Alexe, Wang C. Tan, and Yannis Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. In *VLDB*, 2008.
- [115] Joachim Hammer, Michael Stonebraker, and Oguzhan Topsakal. THALIA: Test Harness for the Assessment of Legacy Information Integration Approaches. In *ICDE*, 2005.
- [116] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The iBench Integration Metadata Generator. Technical report, University of Toronto, 2015.

- [117] Philip A. Bernstein, Todd J. Green, Sergey Melnik, and Alan Nash. Implementing Mapping Composition. *VLDB Journal*, 17(2), 2008.
- [118] Cong Yu and Lucian Popa. Semantic Adaption of Schema Mappings When Schemas Evolve. In *VLDB*, 2005.
- [119] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caulfield. TPC-DI: The First Industry Benchmark for Data Integration. In *VLDB*, 2014.
- [120] Marco Console and Maurizio Lenzerini. Data Quality in Ontology-Based Data Access: The Case of Consistency. In *AAAI*, 2014.
- [121] Meghyn Bienvenu and Riccardo Rosati. Query-based Comparison of OBDA Specifications. In *DL*, 2015.
- [122] Patrick Westphal, Claus Stadler, and Jens Lehmann. Quality Assurance of RDB2RDF Mappings. Technical report, University of Leipzig, 2014.
- [123] Jianing Wang. *A Framework and Architecture for Quality Assessment in Data Integration*. PhD thesis, Birkbeck College, University of London, 2012.
- [124] Martha Impraliou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking Ontology-based Query Rewriting Systems. In *AAAI*, 2013.
- [125] Jose Mora and Oscar Corcho. Towards a Systematic Benchmarking of Ontology-Based Query Rewriting Systems. In *ISWC*, 2014.
- [126] Davide Lanti, Martín Rezk, Mindaugas Slusnys, Guohui Xiao, and Diego Calvanese. The NPD Benchmark for OBDA Systems. In *SSWS*, 2014.
- [127] Joerg Schoenfish and Heiner Stuckenschmidt. Towards Large-Scale Probabilistic OBDA. In *SUM*, 2015.
- [128] Christoph Pinkel, Carsten Binnig, Ernesto Jimenez-Ruiz, Wolfgang May, Dominique Ritze, Martin G. Skjæveland, Alessandro Solimando, and Evgeny Kharlamov. RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In *ESWC*, 2015.
- [129] Patricia C. Arocena, Mariana D’Angelo, Boris Glavic, and Renée J. Miller. iBench First Cut. Technical report, University of Toronto, 2015.
- [130] Zohra Bellahsene, Angela Bonifati, Fabien Duchateau, and Yannis Velegarakis. On Evaluating Schema Matching and Mapping. In *Schema Matching and Mapping*. Springer, 2011.

- [131] Christoph Pinkel, Carsten Binnig, Peter Haase, Clemens Martin, Kunal Sengupta, and Johannes Trame. How to Best Find a Partner? An Evaluation of Editing Approaches to Construct R2RML Mappings. In *ESWC*, 2014.
- [132] Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Towards Approaches for Generating RDF Mapping Definitions. In *ISWC (Posters & Demos)*, 2015.
- [133] Christoph Pinkel. Interactive Pay as You Go Relational-to-Ontology Mapping. In *ISWC, Part II*, 2013.
- [134] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1), 2011.
- [135] Thomas Hornung and Wolfgang May. Experiences from a TBox Reasoning Application: Deriving a Relational Model by OWL Schema Analysis. In *OWLED Workshop*, 2013.
- [136] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics-Doklady*, 10(8), 1966.
- [137] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *ISWC*, 2002.
- [138] Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez-Ruiz, Davide Lanti, Christoph Pinkel, Martín Rezk, Martin G. Skæveland, Evgenij Thorstensen, Guohui Xiao, Dmitriy Zheleznyakov, Eldar Bjørge, and Ian Horrocks. Ontology Based Access to Exploration Data at Statoil. In *ISWC*, 2015.
- [139] Evgeny Kharlamov, Sebastian Brandt, Martin Giese, Ernesto Jiménez-Ruiz, Steffen Lamparter, Christian Neuenstadt, Özgür L. Özçep, Christoph Pinkel, Ahmet Soylu, Dmitriy Zheleznyakov, and Ian Horrocks. Semantic Access to Siemens Streaming Data: The Optique Way. In *ISWC (Posters & Demos)*, 2015.
- [140] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Christoph Pinkel. Evaluating Ontology Alignment Systems in Query Answering Tasks. In *ISWC (Posters & Demos)*, 2014.
- [141] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. OWL: Yet to Arrive on the Web of Data? In *LDOW*, 2012.
- [142] Martin Giese, Ahmet Soylu, Guillermo Vega-Gorgojo, Arild Waaler, Peter Haase, Ernesto Jiménez-Ruiz, Davide Lanti, Martín Rezk, Guohui Xiao, Özgür L. Özçep, and Riccardo Rosati. Optique – Zooming In on Big Data Access. *IEEE Computer*, 48(3), 2015.

- [143] Ondrej Svab, Vojtech Svatek, Petr Berka, Dusan Rak, and Petr Tomasek. OntoFarm: Towards an Experimental Collection of Parallel Ontologies. In *ISWC (Posters & Demos)*, 2005.
- [144] Wolfgang May. Information Extraction and Integration with FLORID: The MONDIAL Case Study. Technical report, Universität Freiburg, Institut für Informatik, 1999.
- [145] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. Large-scale Interactive Ontology Matching: Algorithms and Implementation. In *ECAI*, 2012.
- [146] Alessandro Solimando, Ernesto Jiménez-Ruiz, and Giovanna Guerrini. Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings. In *ISWC*, 2014.
- [147] Freddy Priyatna, Oscar Corcho, and Juan F. Sequeda. Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph. In *WWW*, 2014.
- [148] Heiko Paulheim, Sven Hertling, and Dominique Ritze. Towards Evaluating Interactive Ontology Matching Tools. In *ESWC*, 2013.
- [149] Evgeny Kharlamov, Ernesto Jimenez-Ruiz, Christoph Pinkel, Martín Rezk, Martin G. Skæveland, Ahmet Soyly, Guohui Xiao, Dmitriy Zheleznyakov, Martin Giese, Ian Horrocks, and Arild Waaler. Optique: Ontology-Based Data Access Platform. In *ISWC (Posters & Demos)*, 2015.
- [150] Peter Haase, Ian Horrocks, Dag Hovland, Thomas Hubauer, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Johan W. Klüwer, Christoph Pinkel, Riccardo Rosati, Valerio Santarelli, Ahmet Soyly, and Dmitriy Zheleznyakov. Optique System: Towards Ontology and Mapping Management in OBDA Solutions. In *WoDOOM*, 2013.
- [151] Peter Haase, Michael Schmidt, and Andreas Schwarte. The Information Workbench as a Self-Service Platform for Linked Data Applications. In *COLD*, 2011.
- [152] Andreas Schultz, Andrea Matteini, Robert Isele, Christian Bizer, and Christian Becker. LDIF - Linked Data Integration Framework. In *COLD*, 2011.
- [153] Tomáš Knap, Maria Kukhar, Bohuslav Macháč, Petr Škoda, Jiří Tomeš, and Ján Vojt. UnifiedViews: An ETL Framework for Sustainable RDF Data Processing. In *ESWC (Posters & Demos)*, 2014.

- 
- [154] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and Maintaining Links on the Web of Data. In *ISWC*, 2009.
- [155] Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Dmitry Zheleznyakov, and Ian Horrocks. OptiqueVQS: Towards an Ontology-based Visual Query System for Big Data. In *MEDES*, 2013.