

Jérôme Euzenat
Pavel Shvaiko

Ontology Matching

Second Edition



Springer

Ontology Matching

Jérôme Euzenat • Pavel Shvaiko

Ontology Matching

Second Edition



Springer

Jérôme Euzenat
INRIA and LIG
Grenoble, France

Pavel Shvaiko
Informatica Trentina SpA,
while at Department of Engineering
and Computer Science (DISI),
University of Trento,
while at Web of Data, Bruno Kessler
Foundation - IRST
Trento, Italy

ISBN 978-3-642-38720-3
DOI 10.1007/978-3-642-38721-0
Springer Heidelberg New York Dordrecht London

ISBN 978-3-642-38721-0 (eBook)

Library of Congress Control Number: 2013952732

© Springer-Verlag Berlin Heidelberg 2007, 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

*À mes parents et grand-parents qui ont poussé derrière,
à Anton et Johanna qui tirent par devant,
et à Jutta qui marche à mes côtés.*

Jérôme

Моим родителям, Парисе и Леониду,
за понимание и поддержку.
*A Marlene, il mio amore, che mi è sempre stata vicino.
To Tomas and Emma for cheering up and joyful smiles.*

Pavel

Preface

An ontology typically provides a vocabulary describing a domain of interest and a specification of the meaning of terms in that vocabulary. Depending on the precision of this specification, the notion of ontology encompasses several data or conceptual models, e.g., classifications, database schemas, fully axiomatised theories. Ontologies tend to be everywhere. They are viewed as the silver bullet for many applications, such as database integration, peer-to-peer systems, e-commerce, semantic web services, social networks (Fensel 2004). They are, indeed, a practical means to conceptualise what is expressed in a computer format (Brodie et al. 1984; Guarino 2009). However, in open or evolving systems, such as the semantic web, different parties would, in general, adopt different ontologies. Thus, merely using ontologies, like using XML, does not reduce heterogeneity: it raises heterogeneity problems to a higher level.

For instance, imagine two organisations dealing with books: one is a cultural product electronic commerce site (which sells books, music, movies, etc.) and the other is a university library. Both organisations deal with some related products, the books, but are concerned with different aspects of these: the seller is concerned by the margin, the publisher or the type of binding; the library, in turn, pays more attention to the topic, the size and the year of publication. Both are concerned by the price and the author. Yet they may consider these differently, because the price can include tax and shipping fees or not and being expressed in different currencies or because the authors can be denoted by individual objects or by the character string of their names. Moreover, the seller may organise books according to their commercial types and the library according to their literary types. In summary, these two organisations will obviously have different and heterogeneous ontologies.

The book seller and the library may have to interact, for example, because the latter wants to order books to the former or because the former wants to digitise the collections of the latter. In order to do so seamlessly, they need to find the correspondences between the entities in their respective ontologies. The correspondences may express that what is called a book in the ontology of the seller stands for what is called a volume in that of the library. Furthermore, the price in the seller ontology should be multiplied by a tax rate for obtaining the corresponding price in the

library ontology. The process of finding these correspondences is called ‘ontology matching’.

This book is devoted to ontology matching as a solution to the semantic heterogeneity problem faced by computer systems. Ontology matching aims at finding correspondences between semantically related entities of different ontologies. These correspondences may stand for equivalence as well as other relations, such as consequence, subsumption, or disjointness, between ontology entities. Ontology entities, in turn, usually denote the named entities of ontologies, such as classes, properties or individuals. However, these entities may also be more complex expressions, such as formulas, concept definitions, queries or term building expressions. Ontology matching results, called alignments, can thus express with various degrees of precision the relations between the ontologies under consideration.

Alignments can be used for various tasks, such as ontology merging, query answering, data translation or for browsing the semantic web. In the above mentioned example, the library can take advantage of alignments for automatically ordering a book and the seller can use them for checking the availability of a reference by the library. Matching ontologies enables the knowledge and data expressed in the matched ontologies to interoperate. It is thus of utmost importance for the above mentioned applications whose interoperability is jeopardised by heterogeneous ontologies.

Many different matching solutions have been proposed so far from various viewpoints, e.g., databases, information systems, artificial intelligence. They take advantage of various properties of ontologies, e.g., structures, data instances, semantics, or labels, and use techniques from different fields, e.g., statistics and data analysis, machine learning, automated reasoning, and linguistics. These solutions share some techniques and tackle similar problems, but differ in the way they combine and exploit their results. As a consequence, they are quite difficult to compare and describe, lacking a uniform framework.

About *Ontology Matching*

Ontology Matching aims at being a reference book that presents currently available work in the topic in a uniform framework. In particular, though we use the word ontology, the work and the techniques considered in this book can equally be applied to database schema matching, catalogue integration, XML schema matching and other related problems. The objectives of the book include presenting (i) the state of the art and (ii) the latest research results in ontology matching by providing a detailed account of matching techniques and matching systems in a systematic way from theoretical, practical and application perspectives. The main emphasis of this book is thus on technical solutions for matching.

We have aimed at a sufficiently comprehensive and documented book so that readers can find and learn about almost any subject related to ontology matching and be referred to further reading. Several topics are not covered in full depth but presented only in some salient details for completeness purpose.

It is not the goal of this book to advocate one approach to ontology matching against the others, but rather to show the variety of approaches and their adequacy in different contexts. We are convinced that there is not one unique approach to ontology matching. We concentrate, however, on automatic solutions for matching. Many applications require submitting matching results to user scrutiny and control before using them, but the better the automated part of the task, the easier the control.

This book provides a comprehensive coverage of ontology matching for the researcher and the practitioner. In particular, it reconsiders former frameworks and classifications, broadening their scope and accounting for more solutions. It goes as far as describing in detail basic techniques used in matching systems, reviewing available systems, providing a framework for their evaluation and discussing their applications. This unified view of ontology matching techniques and solutions aims at being the starting point to implementing matching solutions dedicated to a particular application context or developing new techniques. So readers should find in this book inspiration for implementing and understanding matching, they should not expect the ultimate matching solution to be unveiled.

Ontology Matching is not meant to be a textbook, though it features exercises for a selected number of chapters. These exercises can help readers in evaluating their understanding of some technical concepts. This book is also complemented by a web site¹ which features additional information and resources.

Novelty of the Second Edition

Six years have passed since the first edition of *Ontology Matching*, during which the field has made considerable progress Shvaiko and Euzenat (2013). Although, this did not affect the relevance of the first edition, we felt the need to update its content and introduce several novel topics.

A new trend that has risen during the few past years is *linked data* and the subsequent need for data interlinking. Data interlinking falls technically under the definition that we gave for ontology matching and shares enough similarity, so we address this topic to a certain extent (see, in particular, Sects. 1.3, 5.4.2, and 12.4). In fact, ontology matching and data interlinking can be used for improving each other. However, we think that the subject deserves a full autonomous treatment as such.

The new Chap. 3 provides methodological guidelines for people wanting to start a project involving ontology matching. This chapter introduces the alignment life cycle and presents the articulation of the various techniques presented in this book.

Due to the development of matching techniques, the former Chaps. 4 ('Basic techniques') and 5 ('Matching strategies') have been reorganised into three chapters: Chap. 5 covering most of the former Chap. 4 and concerned with local comparison measures, Chap. 6 gathering matching methods working globally on ontologies,

¹<http://book.ontologymatching.org>.

which were dispatched in the two former chapters, and Chap. 7 covering most of its former counterpart. The proposed ‘Classification of ontology matching techniques’ (Chap. 4) has also been revised in light of new techniques. The respective number of matching systems and frameworks overviewed in Chaps. 8 and 10 increased from 50 to over 100.

We renamed Chap. 11, ‘Explaining alignments’, into ‘User involvement’. This is to account for the various ways in which users can participate in the matching task beyond being simply explained the output of the process.

Finally, several topics have been worth a new specific section due to their importance in the current matching theory and practice, such as partitioning and pruning (Sect. 7.1.1), context-based matching (Sect. 7.3), matcher tuning (Sect. 7.6), and alignment metadata (Sect. 10.2), to mention a few.

Outline of the Book

This book is organised in five parts.

Part I is dedicated to the motivation and the definition of the ontology matching problem. The motivation is given in Chap. 1 through the presentation of various applications that can take advantage of matching ontologies and the presentation of how matching contributes to these applications. In Chap. 2, the ontology matching problem is technically defined in various instances of ontology matching occurring in different contexts, such as folksonomies, classifications, databases, XML and entity–relationship schemas and finally formal ontologies. It justifies the emphasis of this book on ontology matching and provides definitions for the vocabulary used. Finally, it technically defines the ontology expression languages, the ontology matching process and its result: the alignment. Chapter 3 provides methodological guidelines for carrying out an ontology matching project through the whole alignment life cycle: from matching ontologies to evolving alignments. It articulates most of the remaining chapters in a rational process plan.

Part II provides a comprehensive coverage of the techniques currently used for ontology matching. It is the main part of the book. Chapter 4 defines a classification of matching approaches. Chapter 5 presents the basic similarity or dissimilarity measures that can be used for comparing ontology entities. These techniques are the basis of most ontology matchers. Chapter 6 discusses more elaborate techniques, which match ontologies by comparing them globally. This may involve propagating similarities globally, from basic measures, to reach an equilibrium. The composition of ontology matching systems from these techniques is considered in Chap. 7, which presents techniques that do not perform matching themselves, but rather manipulate matchers and alignments.

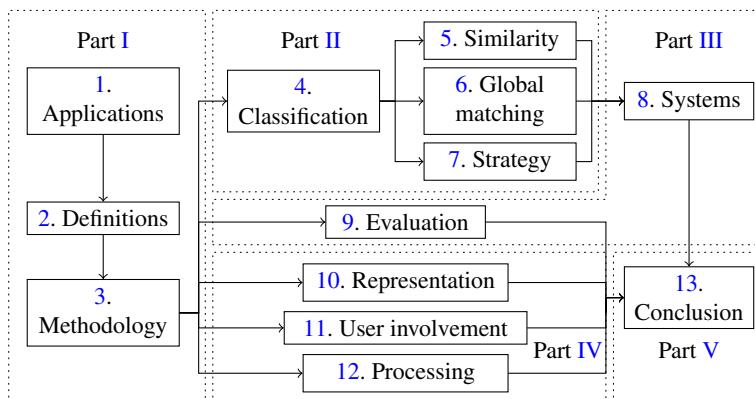
Part III is devoted to packaged matching systems that can be manipulated in applications. Chapter 8 presents a large panel of state-of-the-art matching systems. The reader will find that the basic techniques presented before can lead to a large diversity of systems. Chapter 9 is dedicated to the evaluation of matching solutions.

It presents techniques for discriminating empirically among these systems and evaluating their suitability to a particular application.

Part IV is devoted to the use of the ontology matching results in applications once they have been obtained. Chapter 10 considers how alignments can be expressed either for being stored or for being communicated between systems. This chapter also presents frameworks in which alignments may be both obtained and used in various ways. Chapter 11 deals with user involvement. This is important when matching is not expected to be automatic. Finally, Chap. 12 addresses the ultimate use of ontology matching results through their implementation as effective procedures, e.g., rules, articulation axioms, mediators that can be used within applications.

Part V concludes the book, summarising the current state of ontology matching and emphasising remaining problems that will have to be addressed by further research.

A graphical representation of this organisation is presented below. The arrows offer different independent reading paths through the book.



Readership and Lecture Guide

This book is intended for researchers and practitioners of information and ontology engineering.

The book outline provides a progressive presentation of the ontology matching field and can be read in its entirety. However, each chapter considers ontology matching under a different perspective and can be read in isolation (though it is advised to read the first part before any other). Those who are only interested in getting acquainted with ontology matching can start by reading Chaps. 1, 2 and 13.

For researchers and students dealing with the problem of semantic heterogeneity, we provide not only a comprehensive overview of the state of the art in ontology matching, but also present in detail recent research developments. They show how

ontology matching technologies are going to evolve, indicating which research topics are in the academic agenda and which of them represent the scientific challenges. A course on ontology matching should take some motivations from Chap. 1, explain the concepts introduced in Chap. 2, get inspiration from the guidelines presented in Chap. 3, use the classification of Chap. 4 for exposing Chaps. 5, 6 and 7 and certainly provide some insights from Chap. 9.

For information technology practitioners, both from industry and academia, who want to implement an ontology matching component, this book will help take advantage of state-of-the-art solutions. These readers will take more profit in Chaps. 5, 6, 7, 8, 10, 11 and 12.

For professionals in the areas of e-commerce and knowledge management, the book provides decision support on the use of ontology matching technologies, information about potential problems, and guidelines for the successful application of existing approaches. These readers will take more profit in Chaps. 1, 2, 3, 4, 8, 9, 10 and 12.

We only expect from readers a basic knowledge about data and conceptual modelling and graph theory. Knowledge about logics can also be helpful, though not strictly necessary.

Acknowledgements

The work presented in this book has been partly supported by the Knowledge Web network of excellence (IST-2004-507482) of the European Commission 6th Framework Programme for Research and Technological Development. We emphasise the crucial role played by the European networks of excellence in providing support for cooperative research on important and emerging topics such as this one. This book testifies to the rich working atmosphere these networks contributed to create.

We thank all the participants of the Heterogeneity workpackage of Knowledge Web and, in particular, Than-Le Bach, Jesus Barrasa, Paolo Bouquet, Jan De Bo, Jos De Bruijn, Rose Dieng-Kuntz, Enrico Franconi, Raúl García Castro, Manfred Hauswirth, Pascal Hitzler, Mustafa Jarrar, Markus Krötzsch, Ruben Lara, Małgorzata Mochol, Amedeo Napoli, Luciano Serafini, François Sharffe, Giorgos Stamou, Heiner Stuckenschmidt, York Sure, Vojtěch Svátek, Valentina Tamma, Sergio Tessaris, Paolo Traverso, Raphaël Troncy, Sven van Acker, Frank van Harmelen, and Ilya Zaihrayeu.

Some people had a specific impact on the book through many fruitful discussions, detailed technical feedback on various ontology matching themes, joint work and continuous support during the time we have been elaborating on it. We are very grateful for this to Marc Ehrig, Fausto Giunchiglia, Chan Le Duc, Loredana Laera, Angela Locoro, Diana Maynard, Deborah McGuinness, Petko Valchev, Mikalai Yatskevich, and Antoine Zimmermann.

We also thank Amedeo Napoli for his careful reading, as well as Vincent Englebert, Janina Fengel, Jike Ge, Karl Hammar, Mahboobeh Houshmand, Wei Hu,

Sajjad Hussain, Antoine Isaac, Jun Liang, Tobias Rafreider, Tuukka Ruotsalo, Juan Sequeda, Ondřej Šváb-Zamazal, Sinan Yurtsever, and Antoine Zimmermann for pointing out mistakes in the first version of this book. We are indebted to Fiona McNeill for the time she kindly spent on a first complete draft of this book and her insightful suggestions. Jérôme specially thanks Fred Freitas for hosting him at Porto de Galinhas to quietly work on the second edition of the book.

Finally, we are grateful to our Springer Verlag editor, Ralf Gerstner, for his belief that we had material for such a book and for his kind patience during its production.

Grenoble, France

Jérôme Euzenat

Trento, Italy

Pavel Shvaiko

Contents

Part I The Matching Problem

1 Applications	3
1.1 Ontology Engineering	3
1.2 Information Integration	5
1.3 Linked Data	11
1.4 Peer-to-Peer Information Sharing	13
1.5 Web Service Composition	16
1.6 Autonomous Communication Systems	18
1.7 Navigation and Query Answering on the Web	20
1.8 Summary	23
2 The Matching Problem	25
2.1 Vocabularies, Schemas and Ontologies	25
2.2 Ontology Language	33
2.3 Types of Heterogeneity	37
2.4 Terminology	39
2.5 The Ontology Matching Problem	41
2.6 Summary	53
3 Methodology	55
3.1 The Alignment Life Cycle	56
3.2 Identifying Ontologies and Characterising Needs	57
3.3 Retrieving Existing Alignments	60
3.4 Selecting and Composing a Matcher	61
3.5 Matching Ontologies	63
3.6 Evaluating Alignments	64
3.7 Enhancing Alignments	66
3.8 Storing and Sharing	67
3.9 Rendering and Processing Alignments	68
3.10 Summary	68

Part II Ontology Matching Techniques

4 Classifications of Ontology Matching Techniques	73
4.1 Matching Dimensions	73
4.2 Classification of Matching Approaches	75
4.3 Classes of Concrete Techniques	79
4.4 Other Classifications	82
4.5 Summary	83
5 Basic Similarity Measures	85
5.1 Similarity, Distances and Other Measures	85
5.2 Name-Based Techniques	87
5.3 Internal Structure-Based Techniques	106
5.4 Extensional Techniques	112
5.5 Summary	120
6 Global Matching Methods	121
6.1 Relational Techniques	121
6.2 Iterative Similarity Computation	130
6.3 Matching as Optimisation	137
6.4 Probabilistic Matching	140
6.5 Semantic Techniques	145
6.6 Summary	148
7 Matching Strategies	149
7.1 Ontology Partitioning and Search-Space Pruning	149
7.2 Matcher Composition	153
7.3 Context-Based Matching	156
7.4 Similarity and Alignment Aggregation	160
7.5 Matching Learning	172
7.6 Matcher Tuning	180
7.7 Alignment Extraction	186
7.8 Alignment Improvement	192
7.9 Summary	196
Part III Systems and Evaluation	
8 Overview of Matching Systems	201
8.1 Schema-Based Systems	203
8.2 Instance-Based Systems	233
8.3 Mixed, Schema-Based and Instance-Based Systems	242
8.4 Metamatching Systems	262
8.5 Summary	269

Contents	xvii
9 Evaluation of Matching Systems	285
9.1 Evaluation Principles	285
9.2 Data Sets for Evaluation	291
9.3 Evaluation Measures	300
9.4 Application-Specific Evaluation	314
9.5 Summary	317
Part IV Representing, Explaining, and Processing Alignments	
10 Frameworks and Formats: Representing Alignments	321
10.1 Alignment Formats	321
10.2 Alignment Metadata	337
10.3 Alignment Frameworks	340
10.4 Summary	350
11 User Involvement	353
11.1 Individual Matching	353
11.2 Collective Matching	357
11.3 Explaining Alignments	360
11.4 Alignment Editors and Visualisers	369
11.5 Summary	375
12 Processing Alignments	377
12.1 Ontology Merging	378
12.2 Ontology Transformation	380
12.3 Data Translation	381
12.4 Data Interlinking	385
12.5 Mediation	387
12.6 Reasoning	389
12.7 Alignment Services and Repositories	390
12.8 Alignment Evolution	394
12.9 Summary	395
Part V Conclusions	
13 Conclusions	399
13.1 A Brief Outlook of the Trends in the Field	399
13.2 Future Challenges	401
13.3 Final Words	404
Appendix A Legends of Figures	407
Appendix B Running Example	409
Appendix C Exercises	423
Appendix D Solutions	431
References	463
Index	497

Part I

The Matching Problem

Chapter 1

Applications

This chapter overviews the major ontology matching applications. These applications are presented from a technological point of view, such as ontology integration or linked data, rather than from an economic sector or a domain of interest point of view, such as biomedicine or electronic government. Ontology matching is an enabling operation spreading all domains of interest (running in heterogeneous settings).

Matching models is an important operation in traditional applications, such as ontology integration, schema integration, or data warehouses. Typically, these applications are characterised by heterogeneous structural models that are analysed and matched either manually or semi-automatically at design time. In such applications, matching is a prerequisite of running the actual system.

A line of applications that can be characterised by their dynamics, e.g., agents, peer-to-peer systems, web services, has emerged. Such applications, contrary to traditional ones, require a run-time matching operation and often take advantage of more explicit conceptual models.

We first present some well-known applications where matching has been recognised as a plausible solution for a long time. These are ontology engineering (Sect. 1.1) and information integration, including schema integration, catalogue integration, data warehouses and data integration (Sect. 1.2). Then, we discuss some recently emerged new applications, such as linked data (Sect. 1.3), peer-to-peer information sharing (Sect. 1.4), web service composition (Sect. 1.5), autonomous communication systems, including agents and mobile devices communication (Sect. 1.6), navigation and query answering on the web (Sect. 1.7). For each class of technological applications we provide a concise motivating scenario from a domain of interest. Finally, the legends to the figures illustrating applications under consideration can be found in Appendix A.

1.1 Ontology Engineering

A context in which users are confronted with heterogeneous ontologies is ontology engineering, and, more generally, the task of designing, implementing and maintain-

ing ontology-based applications. This activity requires ontology matching support because ontology engineering has to deal with multiple, distributed and evolving ontologies. Below we provide a short motivating scenario from the biomedical domain, which is concerned with ontology evolution.

Motivating Scenario from the Biomedical Domain Development of ontologies in the biomedical domain started more than a decade ago. In this context, the most prominent example of ontology development is that of the gene ontology project.¹ This is a collaborative effort to develop and use ontologies to support biologically meaningful annotation of genes and their products in a wide variety of organisms (Gene Ontology Consortium 2008). It covers molecular functions, associated biological processes and cellular components. The gene ontology is maintained through a CVS repository, which traces only syntactic differences; hence, for instance, differences between class versions, splits or merges of concepts are not captured, though such differences may invalidate annotations. These differences may be revealed with the help of matching. Moreover, there exist several infrastructures promoting the development and use of ontologies in biomedicine, such as OBO Foundry (Smith et al. 2007) and BioPortal (Musen et al. 2012). For example, BioPortal offers a repository of more than 270 biomedical ontologies, with support services for annotating, retrieving or performing analytics over data (Sect. 12.7.1).

1.1.1 *Ontology Editing and Import*

Ontology heterogeneity may be first faced while designing an ontology for a domain of interest. Ontology-based system designers often have to integrate different ontologies, either for the sake of enforcing reuse, and thus not multiplying ontologies on the same topic, or because it is necessary for interconnecting various relevant resources.

It is often the case that application engineering requires an external set of ontologies to be put together. For instance, building a library cataloguing ontology may require assembling ontologies for people, books and topics as well as ontologies for measurement units, geographic coordinates, book identification numbers, metadata, etc. These ontologies share related concepts, for instance, the friend-of-a-friend (FOAF²) ontology (which can be used as a starting point for modelling people) offers a Document concept that has to be related to the classes of the book identification numbers ontology.

Ontology engineers need support for (i) identifying the relevant ontologies and (ii) matching and recording the relations between entities in these ontologies. Additionally, they may want to import the identified ontologies and merge them (in

¹<http://www.geneontology.org>.

²<http://www.foaf-project.org>.

which case, they will use some axioms generated from the result of the matching phase) or to use data expressed under another ontology in the application (in which case, they will generate a mediator from the matching result).

The situation under consideration is simple because it is static. In fact, ontologies are encountered at design time and mediators can be built at that moment. Thus, application developers can find correspondences and design the necessary transformations manually. Some ontology engineering tools provide support for finding the correspondences, for example, Protégé through the Prompt suite of tools (Sect. 10.3.5). Newer ontology development environments will have to take into account, from the beginning, the existence of multiple ontologies and the need for mediators between them. For instance, Schemr offers a search engine to look for schemas from a metadata repository, such that schema fragments can be reused, thereby providing an immediate bootstrap path for a modelling issue at hand (Chen et al. 2011).

1.1.2 *Ontology Evolution and Versioning*

It is natural that domains of interest, application requirements and the way in which knowledge engineers conceptualise these by means of ontologies undergo changes and evolve over time. Moreover, ontology development, similar to software development, is often performed in a distributed and collaborative manner. Therefore, multiple versions of the same ontology, e.g., the gene ontology, often exist. Some applications keep their ontologies up to date, while others may continue to use old ontology versions and update them on their own. This situation arises because knowledge engineers and developers usually do not have a global view of how and where the ontologies have changed. In fact, change logs may not always be available (which is often the case in distributed ontology development). Therefore, developers need to manage and maintain the different versions of their ontologies.

The matching operation can help here (see Fig. 1.1). Its main focus is on discovering the differences, e.g., which ontology entities have been added, deleted or renamed, between two ontology versions (Roddick 1995; Noy and Klein 2004; Noy and Musen 2002b, 2004; Kirsten et al. 2011; Hartung et al. 2013).

1.2 Information Integration

Information integration is one of the oldest classes of applications where matching is viewed as a solution. Under the information integration heading, we gather here such problems as schema integration (Batini et al. 1986; Sheth and Larson 1990; Spaccapietra and Parent 1991; Parent and Spaccapietra 1998), data warehousing (Bernstein and Rahm 2000; Dessloch et al. 2008), data integration (also known as enterprise information integration) (Chawathe et al. 1994; Wache et al. 2001;

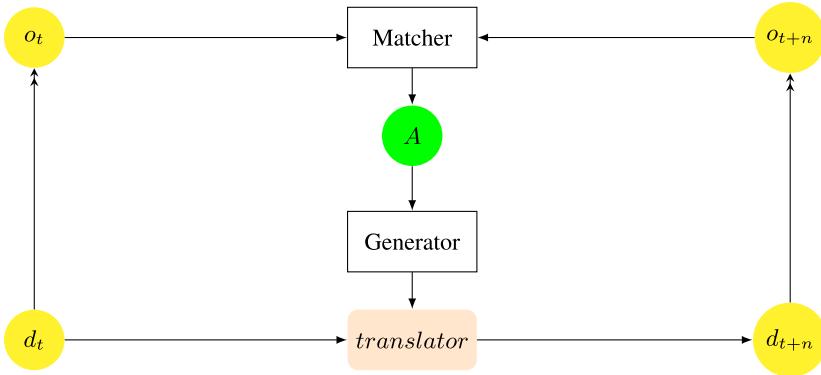


Fig. 1.1 Ontology evolution. In this scenario it is useful to (1) *match* the old version o_t and the new version o_{t+n} of the ontology, thus resulting in an alignment (A) between these versions, (2) *generate* a transformation by using this alignment and (3) *translate* the underlying data instances d_t to d_{t+n} .

Draper et al. 2001; Halevy et al. 2005; Seligman et al. 2010; Doan et al. 2012), and catalogue integration (Agrawal and Srikant 2001; Ichise et al. 2003; Bouquet et al. 2003b; Giunchiglia et al. 2005a). Below we provide a short motivating scenario from the cultural heritage domain, which is concerned with data integration.

Motivating Scenario from the Cultural Heritage Domain A typical situation in the cultural heritage domain consists of having several large thesauri, such as *Iconclass*³ (25 000 entities) and the *Aria*⁴ collection (600 terms) from the Rijksmuseum (Amsterdam). The documents indexed by these thesauri are illuminated manuscripts and masterpieces, i.e., image data. The labels are gloss-like, i.e., sentences or phrases describing the concept, since they have to capture what is depicted on a masterpiece. Examples of labels from *Iconclass* include city-view, and landscape with man-made constructions and earth, world as celestial body. In contrast to *Iconclass*, *Aria* uses simple terms as labels. Examples of these include landscapes, personifications and wild animals. Matching between these thesauri, which may be performed at design time, is required in order to enable integrated access to the masterpieces of both collections. Specifically, alignments can be used as navigation links within a multi-faceted browser to access a collection via thesauri it was not originally indexed with (van Gendt et al. 2006).

A general information integration scenario is presented in Fig. 1.2: given a set of local information sources (local ontologies LO_1, \dots, LO_n) potentially storing their data in different formats, e.g., SQL DDL, XML, or RDF, provide users with a uniform query interface via the mediated (or global) ontology CO , to all the local infor-

³<http://www.iconclass.nl>.

⁴<http://www.rijksmuseum.nl/en/research-and-library>.

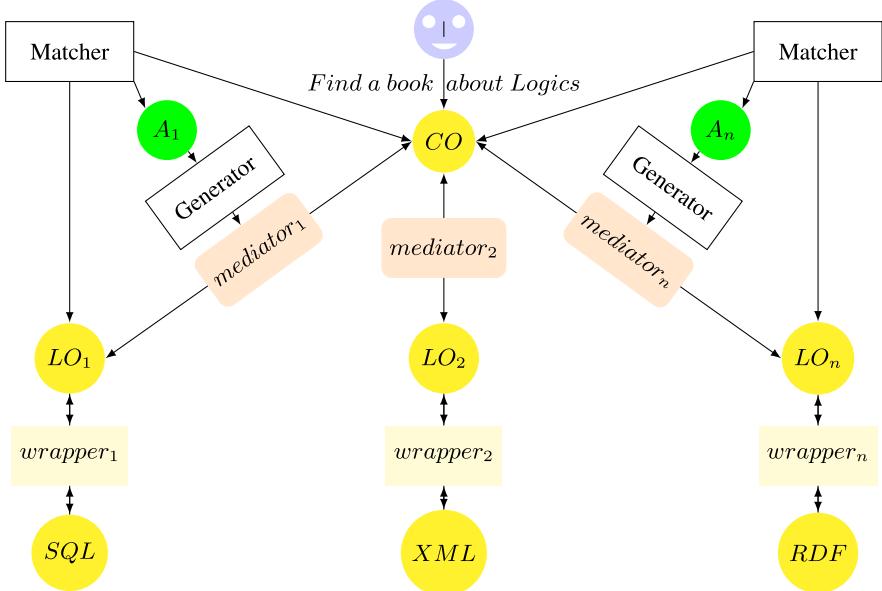


Fig. 1.2 General (centralised) information integration. The data sources (*SQL*, *RDF*, etc.) are wrapped (*wrapper_i*) to ontologies (*LO_i*) which (1) are *matched* against a common ontology (*CO*). The alignments (*A_i*) between these (2) help *generate* mediators (*mediator_i*) which in turn (3) *transform* queries against the common ontology into a query to the information source and (4) the answers in the other way.

mation sources. This allows users to avoid querying the local information sources one by one, and to obtain a result from them just by querying a common ontology.

For example, if users pose queries like *find a book about Logics* to a common ontology, then, an information integration system communicates with information sources, e.g., bookstores or libraries, and returns a reconciled result based on the input provided by these sources. In general, the information integration system performs several macro steps. These include

- interpreting, i.e., rewriting, the query in terms of the common ontology;
- identifying correspondences between semantically related entities of the local information sources and the common ontology;
- translating the relevant data instances of the local information sources, involved in handling the query, into the knowledge representation formalism of the information integration system; and
- reconciling the results obtained from multiple information sources, namely detecting and eliminating, redundancies and duplicates, before returning the final answer.

Identifying the correspondences between semantically related entities of the local information sources and the common ontology is a matching step. We limit our

vision of matching to the description above for the moment; we will expand it to some extent in the next sections.

In some concrete information integration scenarios, the common ontology can be either materialised, i.e., physically existing, or virtual. Below, we discuss these options in more details.

1.2.1 Schema Integration

Schema integration is the oldest scenario (Batini et al. 1986; Sheth and Larson 1990; Parent and Spaccapietra 1998). Assume that two (or more) enterprises want to perform either a merger or an acquisition among them. Ultimately, these enterprises have to integrate their databases into a single one. Usually, a first technical step is to identify correspondences between semantically related entities of the schemas before merging the databases. This step, known as matching, is required even if the databases to be integrated are coming from the same domain of interest, e.g., book selling, car rentals. This may be because the schemas have been designed and developed independently, or because people follow diverse modelling principles and patterns, even if they have to encode the same real-world object. Finally, the schemas to be integrated might have been developed according to different business goals. This makes the matching problem even harder.

Other scenarios may be classified under the schema integration heading, such as (tightly-coupled) federated databases (Sheth and Larson 1990). These typically have one global schema providing a unified access to the federation of component databases. Component databases, in turn, are autonomous. Thus, in this application when, for example, one component schema of the federated database is changed, the federated (global) schema has consequently to be also reconsidered. Matching can help in identifying these changes.

Finally, it is worth noting the applications that we are not discussing here, e.g., distributed databases systems (Özsu and Valduriez 2011). These are usually designed in a centralised way, e.g., by a database administrator, and therefore, semantic heterogeneity does not exist there by construction (Elmagarmid et al. 1999).

1.2.2 Catalogue Integration

In Business-to-Business (B2B) applications, trade partners store information about their products in electronic catalogues. Typical examples of catalogues are product directories of electronic sale portals, such as Amazon or eBay. In order for a merchant to participate in the marketplace, it has to determine correspondences between entries of its catalogues and those of the marketplace catalogue (see Fig. 1.3). This process of finding correspondences among entries of the catalogues is referred to as catalogue matching (Bouquet et al. 2003b). Looking at this problem from a merchant viewpoint, matching has to be performed for each marketplace in which it

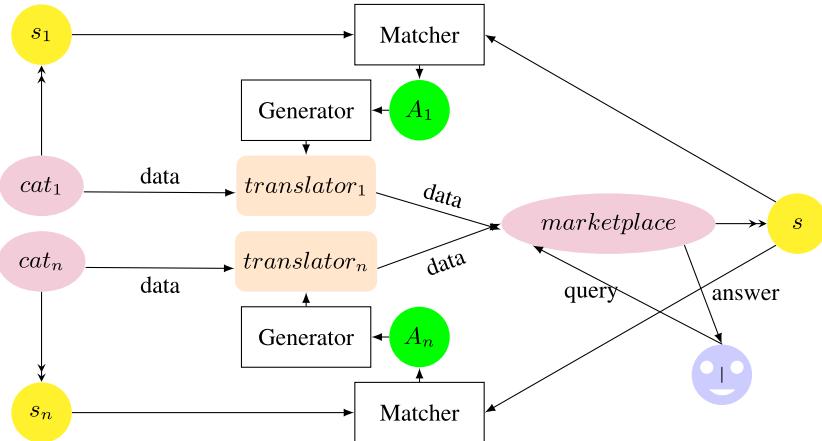


Fig. 1.3 Catalogue integration with matching. Each merchant (1) *matches* its catalogue (s_i) with that of the marketplace (s). From the resulting alignment (A_i), (2) a data translation program ($translator_i$) is *generated* which is (3) *used* for loading the catalogue (cat_i) to the marketplace. Users can ask queries to the marketplace and (4) *receive* answers based on the integrated catalogue.

would like to participate. Having identified the correspondences between catalogue entries, they are further analysed in order to generate query expressions that automatically translate data instances between the catalogues. Finally, having matched the catalogues, users of a marketplace have a unified access to the products which are on sale. Such a situation, involving interactions between marketplaces and merchants, is described in Fig. 1.3. It can be viewed as a typical example of integrating local data sources into a data warehouse, see also (Bernstein and Rahm 2000).

Another catalogue integration scenario deals with (typically large-scale) product classifications, such as UNSPSC⁵ (The United Nations Standard Products and Services Code) and eCl@ss⁶ (Standardised Material and Service Classification). In a sense, this scenario can be thought of as one which enables interoperability among multiple B2B marketplaces, thus, facilitating product exchange between the enterprises subscribing to different product classifications (Schulten et al. 2001). This is achieved by establishing correspondences between semantically related entities of the standardised product classifications, which is a matching operation as well.

1.2.3 Data Integration

Data integration is an approach where integration of information coming from multiple local sources is performed *without* first loading their data into a central ware-

⁵<http://www.unspsc.org>.

⁶<http://www.eclasse.de>.

house (Halevy et al. 2005). This allows interoperation across multiple local sources having access to the up-to-date data. The data integration system has to update the central warehouse of the marketplace, instead of merchants in the catalogue integration situation.

The scenario, depicted in Fig. 1.4, is as follows. First, local information sources participating in the application, e.g., bookstore, library, museum, are identified. Then, a virtual common ontology is built. Queries are posed over the virtual common ontology, and are then reformulated into queries over the local information sources, e.g., such as in cultural heritage scenario outlined in Sect. 1.2. In order to enable semantics-preserving query answering, correspondences between semantically related entities of the local information sources and the virtual ontology have to be established. Establishing these correspondences is known as matching.

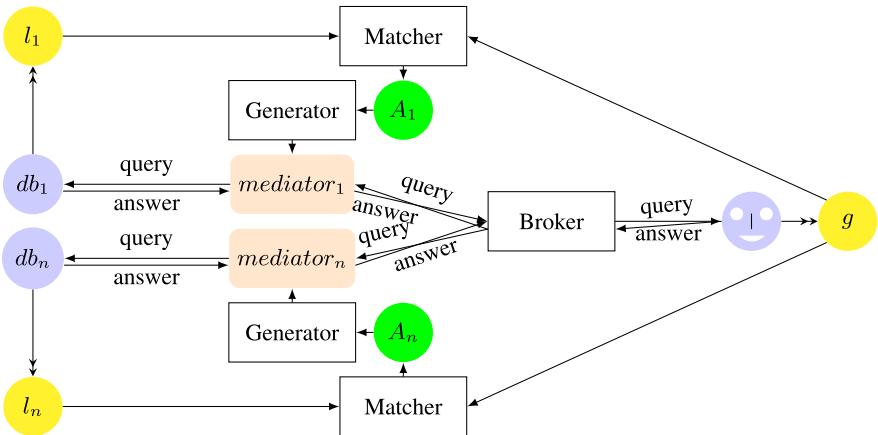


Fig. 1.4 Data integration with matching. Depending on whether the global schema (g) is considered as *matched* against existing local schemas (l_i) or the other way around, this describes the global-as-view or local-as-view approach, respectively. Usually, (1) *matching*, resulting in alignments (A_i), is performed off-line, (2) *generating mediators* ($mediator_i$) for each local database. The query is sent to a *broker* calling the necessary mediators. They (3) *translate* the query, evaluate it against the database and (4) *translate* the answer before returning it.

Query answering is then performed by using these correspondences (mappings) within the Local-as-View (LAV), Global-as-View (GAV), or Global-Local-as-View (GLAV) settings (Lenzerini 2002). In the local-as-view approach, local schemas are defined in terms of the global schema, i.e., mappings are specified by defining each local schema construct as a view over global schema constructs. Queries are processed by means of an inference mechanism that re-expresses the atoms of the global schema in terms of atoms of the local schemas. In global-as-view, a global schema is defined in terms of the local schemas, i.e., mappings are specified by writing a definition of each global schema construct as a view over local schema constructs. Queries are processed by means of unfolding, i.e., by expanding the

atoms according to their definitions (so as to come up with local schema relations). Global-local-as-view is a variation of the local-as-view approach that allows the head of the view definition to contain any query on the local schemas.

Finally, as noticed in (Lenzerini 2002), the main task in these applications is to establish mappings, i.e., to perform the matching operation. Besides using matching results for creating the global (respectively local) views, it can also be used for maintaining them when schemas evolve.

1.3 Linked Data

The web of data is made of data published following the following four principles (Berners-Lee 2009; Heath and Bizer 2011):

1. Resources are identified by URIs.
2. URIs are dereferenceable.
3. When a URI is dereferenced, a description of the identified resource should be returned, ideally adapted through content negotiation.
4. Published web data sets must contain links to other web data sets.

Although not explicitly specified, linked data sources are more usable if they are published with semantic web technologies: URIs for identifying resources, RDF for describing them, OWL for defining the used vocabularies and SPARQL for accessing data. The ‘five stars rating’, to be interpreted as incremental steps, has been introduced for measuring how much of this usability is achieved, namely:

- ★ Publish data on the web in any format, e.g., a scan of a table in PDF, which is typically done through catalogues or portals;
- ★★ Use structured data formats, e.g., a table in Excel instead of its scan in PDF;
- ★★★ Use non-proprietary formats, e.g., CSV instead of Excel, such that users have direct access to the raw data;
- ★★★★ Use universal formats to represent data, such as RDF, which encapsulates both syntax and semantics;
- ★★★★★ Link data to other data sets on the web, thereby providing context.

The first three stars are easy to reach and these already enable some data reuse. However, humans still have to handle all the semantic issues related to integration. In order to have data that is more easily discoverable and interoperable, it is necessary to arrive to the fourth and the fifth stars. This may be achieved by publishing data in RDF, describing the vocabulary in OWL, providing access points in SPARQL and expressing links with the `owl:sameAs` predicate.

Below we provide a short motivating scenario from the Public Sector Information (PSI) domain, which is concerned with enabling data interlinking (the fifth star) between data sets of the linked open data cloud.⁷

⁷<http://lod-cloud.net>.

Motivating Scenario from the Public Sector Information Domain Public Sector Information is viewed as the single largest source of information in Europe. Releasing government information as open, from both legal and technical viewpoints, linked data facilitates transparency and data reuse, which in turn stimulates generation of new business opportunities and favours economic development. The domain is characterised by a variety of information, of variable granularity and quality created by different institutions and represented in heterogeneous formats. Typically, public data is released in the form of catalogues describing the available contents, such as the UK national PSI catalogue of data.gov.uk or the US catalogues of data.gov. An important aspect of opening public information, is its links to the other identical or similar pieces of data available on the web. They increase PSI discoverability, integration, and thus, its reuse (Shadbolt et al. 2011; Shvaiko et al. 2012). The task of data interlinking can be facilitated by matching first the entities of the respective PSI catalogues, thus narrowing down the search space of the possible links.

As mentioned above, one important issue in linked data is to be able to establish links between data sets. The current practice consists of finding, in different data sets, entities representing the same resource and linking their URIs with the `owl:sameAs` construct. Finding such links is reminiscent of entity resolution in natural language processing or record linkage in databases (see Sect. 5.4.2). The main difference with record linkage or deduplication is that it is usually performed within one single database, i.e., all entities are described under the same schema. On the web of data, data providers express data with respect to multiple and heterogeneous schemas or ontologies.

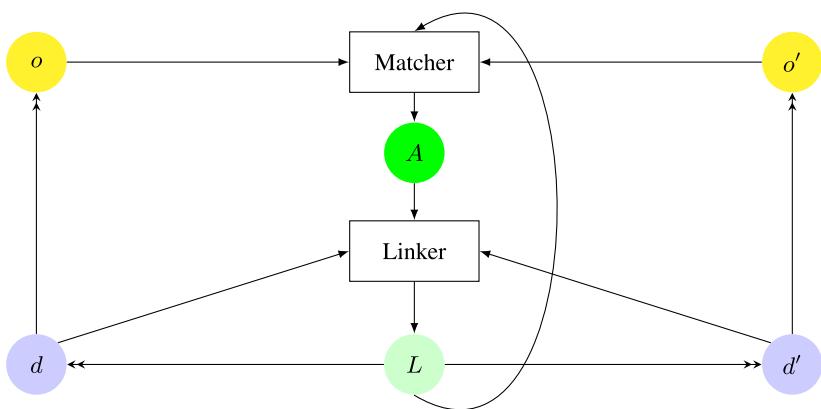


Fig. 1.5 Data interlinking and ontology matching (inspired from (Scharffe and Euzenat 2011)). In linked data, matching is useful to help generating links between different data sets. It starts with (1) *matching* the ontologies (o and o') under which data sets (d and d') are expressed, before (2) using the resulting alignment (A) for *interlinking* the two data sets, i.e., generating `sameAs` relations or links (L) between resources of both data sets. These links can also be efficiently used for (3) *improving* matching.

Data interlinking and ontology matching can be seen as dual operations. On the one hand, when confronted with two data sets using different ontologies, data interlinking can take advantage of ontology matching. On the other hand, ontology matching can implement extensional methods, or instance-based methods, which may take advantage of data interlinking (see Fig. 1.5). Hence, the two processes may be used for reinforcing each other. Matching can start with any of them for contributing to the result of the other, or it may be iterated until their eventual convergence (Scharffe and Euzenat 2011).

1.4 Peer-to-Peer Information Sharing

Peer-to-Peer (P2P) is a distributed communication model in which parties, called peers, have equivalent functional capabilities in providing each other with data and services (Zaihrayeu 2006). P2P networks became popular through the file, e.g., pictures, music, videos, books, sharing paradigm. There exist several widely used P2P file sharing systems, e.g., BitTorrent. These applications describe file contents by a simple schema (set of attributes, such as title of a song, author) to which all peers in the network have to subscribe. These schemas cannot be modified locally by a single peer. Therefore, in the above mentioned systems the semantic heterogeneity problem, at the schema level, does not exist by construction. The use of a single system schema violates the *total autonomy* of peers. Although robust P2P systems allow peers to connect to and disconnect from the network at any time, thereby respecting some forms of peers autonomy, such as *participation autonomy*, they still restrict the *design autonomy* of peers, in matters such as how to describe the data and what constraints apply on the data (Zaihrayeu 2006).

If peers are meant to be totally autonomous, they may use different terminologies and metadata models in order to represent their data, even if they refer to the same domain of interest. Below we provide a short motivating scenario from the emergency response domain, which is concerned with P2P data exchange.

Motivating Scenario from the Emergency Response Domain In a typical emergency situation, such as flooding, or fire in a urban centre, the central control centre routes evacuation information to people in need. However, if this centralised system (or a part of it) breaks down, a backup decentralised system might need to be activated. For example, a bus driver (an institutional peer) evacuating flood victims may need a more detailed map to find an alternative operational road if the followed one is destroyed and the control centre is out of reach. Various geo-agencies (other institutional peers) typically operate on a territory, such as civil protection, urban planning, traffic control, and are usually responsible for providing a specific subset of information for a geographic area and might have the details that the bus driver looks for. Similarly, citizens (volunteer peers) may signal that a certain road is becoming blocked. These non-institutional and institutional peers are expected to establish rapidly a P2P network by forming a community of diverse people engaged

in an emergency. Since there is no a priori knowledge or agreement between peers, on-the-fly matching is needed to automatically make commitments on the vocabulary used between them, e.g., to understand that XMin_BB and XMin refer to the same spatial coverage of a map requested by the bus driver (Vaccari et al. 2009). Thus, matching supports modifying rapidly the emergency plan under the new constraints, such as new information or missing capabilities, based on the best match available at a given time.

1.4.1 Semantic P2P Systems

Semantic P2P systems (Staab and Stuckenschmidt 2006) use more complex specifications of their contents, such as database schemas (Bernstein et al. 2002), or formal ontologies (Rousset et al. 2006), than the classical P2P systems mentioned above. The main idea behind this is to improve search accuracy by providing a finer-grained description of items. For example, users who want to share their book library with their friends may index them by authors, topics, and years of publication. This tagging approach will benefit from using some ontological descriptions, e.g., for retrieving books on mathematics written by Cambridge authors before 1920 as opposed to books by Bertrand Russell in 1908 on logic. For instance, the BibSter system (Haase et al. 2004) uses a bibliographic ontology expressed in RDF.

Systems as BibSter still follow a single ontology approach. More advanced semantic P2P systems relax the homogeneity requirement of classical P2P systems: they allow peers to use independent schemas and ontologies (see Fig. 1.6).

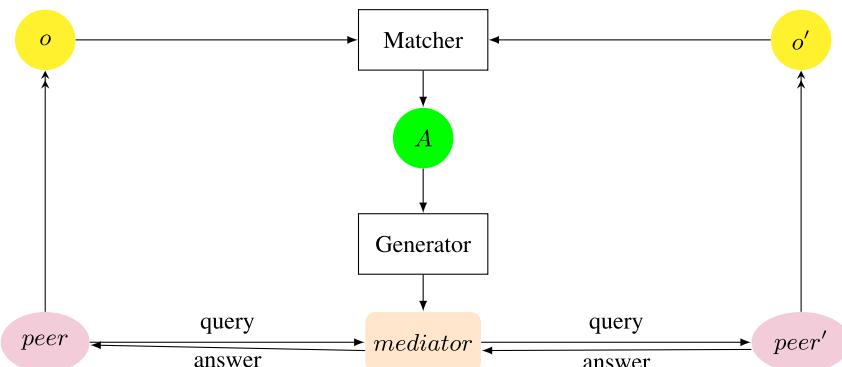


Fig. 1.6 P2P query answering. In this scenario, it is useful to (1) *match* relevant parts of ontologies o and o' , thus resulting in alignment A , (2) *generate* a *mediator* between $peer$ and $peer'$ for translating queries and sometimes for translating answers.

Such applications pose additional requirements on matching solutions. In P2P settings which respect total autonomy of peers, an assumption that all the peers

rely on one global schema, as in data integration, cannot be made because the global schema may need to be updated any time the system evolves (Giunchiglia and Zaihrayeu 2002). While, in the case of data integration, schema matching can be performed at design time, in P2P applications, peers need to coordinate their databases on-the-fly, therefore ultimately requiring run-time schema matching. Finally, incomplete and approximate answers, as long as they are good enough for the application, are also acceptable in such settings. This is the case because some correspondences involved in query answering may become temporarily unavailable or invalid (Shvaiko et al. 2006).

Some examples of P2P data exchange which rely on different peer metadata models, including relational database schemas, XML schemas, RDF schemas, or OWL ontologies are described in (Bernstein et al. 2002; Zaihrayeu 2006; Ives et al. 2004; Nejdl et al. 2002; Rousset et al. 2006; Montanelli and Castano 2008). For example, applications like SomeWhere (Rousset et al. 2006) integrate peer databases and connect them through mappings expressed in Horn clauses from one database to another. When a peer needs to answer a query, the system computes possible expansions of the query with regard to these mappings, i.e., it follows the local-as-view approach (Sect. 1.2.3). Then it sends to each relevant peer the queries that may help answer the initial query and answers are returned and integrated as soon as they arrive. This approach assumes that peer database schemas have been matched off-line beforehand. Thus, only the query answering part of the system takes into account the dynamics of the P2P environment.

1.4.2 Emergent Semantics Between Peers

Emergent semantics (Aberer et al. 2004a, 2004b; Montanelli et al. 2011) is the process by which a set of peers gradually converges towards a consensus ontology through constantly interacting and negotiating the meaning of the terms they use (Atencia and Schorlemmer 2012). This process mimics to some extent the one exhibited by a society of humans and may never reach an end but at least it improves discourse understanding. Since consensus is built incrementally, emerging from different local point-to-point peer agreements, an alignment between ontologies of peers is viewed as a practical means for establishing those local agreements. Thus peers will have to constantly update the relations between their ontologies. These updates can be achieved by matching. The process of emerging semantics between two peers is illustrated in Fig. 1.7.

Constantly matching ontologies can trigger the confrontation and revision of these ontologies themselves. In fact, users may want to establish more consensual ontologies from this confrontation (Zhdanova et al. 2005). There are several ways in which alignments may help:

- Alignments provide a basis from which the negotiation between peers can start (like agent protocols for arguing about correspondences, see Sect. 1.6.1).

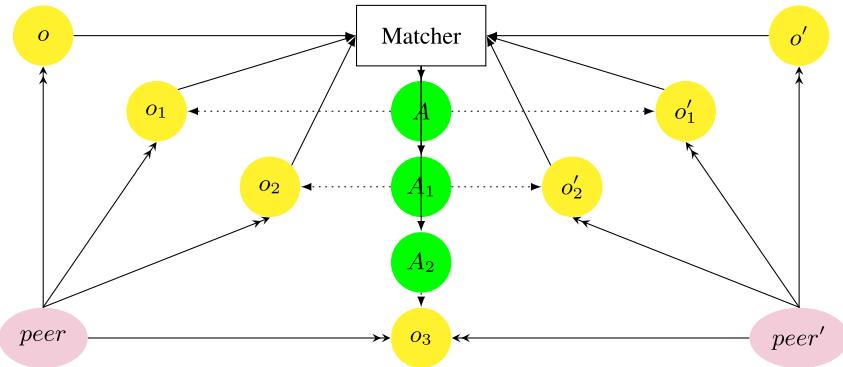


Fig. 1.7 Peer-to-peer and emergent semantics: after a first *matching* between ontologies o and o' , the resulting alignment A causes (dotted line) the peers (*peer* and *peer'*) to evolve their ontologies into o_1 and o'_1 , respectively. In turn, these ontologies (o_1 and o'_1) can again be *matched*, thus resulting in alignment A_1 , and so on and so forth. Ultimately, the peers may converge to a common ontology (o_3).

- Matching algorithms are very often able to compute a distance between ontologies. This is useful when, for instance, a peer wants to find the ‘closest’ ontology.
- By building a network of ontologies together with alignments between them and by exploiting, with the help of social network analysis techniques, the distance between the ontologies, it is also possible to determine the proximity between users or agents (Jung and Euzenat 2007; Atencia et al. 2011). This, in turn, facilitates customising the query answering process, and even consensus building.

These results will help users and communities in consolidating their ontologies by gradually achieving agreements with similar domain representations as well as for determining the most central ontology (in social network analysis terms) for the domain of interest.

1.5 Web Service Composition

Web services are processes that expose their interfaces to the web so that users can invoke them. Semantic web services provide a richer and more precise way to describe services through the use of knowledge representation languages and ontologies (Fensel et al. 2007). Web service discovery and integration is the process of finding a web service able to deliver a particular service and composing several services in order to achieve a particular goal (Paolucci et al. 2002; Medjahed and Bouguettaya 2005; Oundhakar et al. 2005; Algergawy et al. 2010). Below we provide a short motivating scenario from the e-commerce domain, which is concerned with adapter development for web service integration.

Motivating Scenario from the E-commerce Domain A typical situation in electronic commerce is that of service adaptation for handling shopping carts (Nezhad et al. 2010). For example, XWebCheckOut⁸ offers management of the standard shopping cart checkout process to client applications, involving basket, shipping, billing, credit card, and receipt, which are to be integrated within a merchant web site. Similar functionalities are also offered by Google Wallet.⁹ Functionally equivalent or closely similar services have to be compatible and replaceable to facilitate business process automation. However, these services are using different interface and protocol specifications. Interfaces define the set of operations as well as message formats and data types, while protocols define the order in which the operations can be called. So, message names and types can be different as well as message exchange flow can be different. For example, it should be identified that AddOrderRequest is a corresponding message to PlaceOrder. This is a web service interface matching task.

As mentioned previously, web services have been designed for being independent and replaceable. So web service processors are able to incorporate new services in their workflows, and therefore customers can dynamically choose new and more promising services. For that purpose, they must be able to compare the descriptions of these services, in order to know if they are indeed relevant, and to route the knowledge they process, in order to compose different services by routing the output of some service to the input of another service.

However, in the case of semantic web services, which are described with regard to ontologies, imposing a central common ontology (like in single ontology P2P systems), as real-world experiences demonstrate, is not realistic and would freeze the evolution of such services. Henceforth, both for finding the adequate service and for interfacing services, a data mediator comes into play as a bridge between different vocabularies (Bussler et al. 2002; Fensel et al. 2007). From the correspondences between the terms of the descriptions, mediators can translate the output of one service into a suitable input for another service (see Fig. 1.8).

Thus, the core part of a mediator definition is an alignment between two ontologies. This, in turn, can be provided through matching the corresponding ontologies either off-line when someone is designing a preliminary service composition, or dynamically (on-line), when new services are sought for completing a request (Giunchiglia et al. 2006b; Robertson et al. 2006).

For instance, assume that an on-line library service provides its output description in some ontology and a parcel shipping service uses a second ontology for describing its input. Matching these ontologies is useful for: (i) checking that what is delivered by the first service, e.g., a Book, matches what is expected by the second one, e.g., an Object, (ii) verifying preconditions of the second service, e.g., size in centimetres against dimensions in inches, and (iii) generating a mediator able to transform the output of the first service in order to be input to the second one (see Fig. 1.8).

⁸<http://www.programmableweb.com/api/xwebcheckout/>.

⁹<https://developers.google.com/commerce/wallet/>.

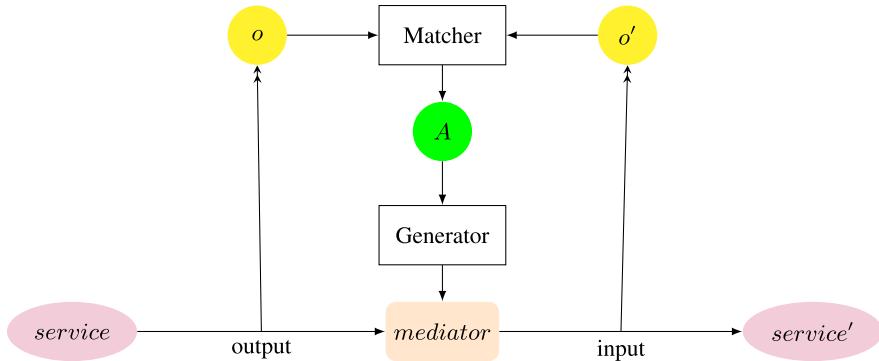


Fig. 1.8 Web service composition. In this scenario it is useful to (1) *match* relevant parts of ontologies o and o' used in service descriptions, thus resulting in alignment A , (2) *generate* a *mediator* between $service$ and $service'$ in order to enable transforming the actual data.

1.6 Autonomous Communication Systems

Other kinds of applications also involve autonomous entities that can meet on a network and which have been designed independently. When these entities are software programs, they have been considered as agents for a long time (Sect. 1.6.1). However, if they are a combination of hardware and software they are a matter of ambient computing (Sect. 1.6.2). Obviously, as we have already discussed in previous sections, such entities cannot share a common ontology. Thus, if they want to communicate, it is useful to match their ontologies. Below we provide a short motivating scenario from the urban services domain, which is concerned with location-based services.

Motivating Scenario from the Urban Services Domain Nowadays, traditional paper maps are progressively replaced by digital maps and location-based services. This facilitates reuse of such digital information and services. However, this reuse is hindered by heterogeneity of the respective (ontological) descriptions, which are typically defined as partial abstraction views of the reality. Urban infrastructures, such as highways or buildings, already report through sensors their states and activities, thereby providing the respective real-time and location-based information. Environmental information, e.g., on wind or humidity, is also acquired through sensors on a large scale, e.g., at city scale, and is usually maintained by different organisations. Urban services, such as on traffic conditions, environmental status, e.g., air quality, energy consumption, can be delivered on a personal device, e.g., smartphone or tablet. They can be improved by making them more personalised, efficient or precise depending on the context. Hence, vast amounts of urban data (streams) should be constantly discovered and processed. The ultimate goal of such services is to create more efficient urban systems, so some faults can be predicted in advance or some resource consumption can be reduced. They will also provide more accurate information to citizens and decision makers. Ontology matching has a role to play

in this context because data streams are heterogeneous and the more precise they are, the more likely it is that they use different ontologies.

1.6.1 *Multiagent Communication*

Agents are software entities characterised by their autonomy and capacity of interaction. They are often divided into cognitive agents and reactive agents. Reactive agents implement a simple behaviour and the strength of these agents is their capacity to let a global behaviour emerge from the individual behaviour of many such agents. Cognitive agents have a rather more elaborate behaviour often characterised as the ability to pursue goals, to plan their actions and to negotiate with other agents in order to achieve their goals.

Agents communicate by exchanging messages expressed in an agent communication languages, such as the FIPA Agent Communication Language (FIPA 2002a, 2002b). These languages determine the ‘envelope’ of the messages and enable agents to position themselves within a particular interaction context. However, they do not specify the actual content of the message, which is often expressed with respect to some ontology accessible to the agent. Current standards for expressing these messages provide slots for declaring the content language and the ontology used.

As a consequence, when two autonomous and independently designed agents meet, they have the opportunity to exchange messages but little chance to understand one another if they do not share the same content language and ontology. It is thus useful to help these agents to match their ontologies in order to either translate their messages or integrate bridge axioms in their own models. Several proposals have been made to assess the correspondences between the terms of the ontologies (van Eijk et al. 2001; Wiesman et al. 2002; Bailin and Truszkowski 2002; Wang and Gasser 2002; Euzenat et al. 2005).

Agents confronted with heterogeneous ontologies have to find the correspondences between these ontologies in order to start understanding each other’s messages. They can perform ontology matching by themselves or by taking advantage of alignment libraries or matching services. Once an alignment is obtained, agents can start a negotiation phase (Laera et al. 2006; Trojahn et al. 2011) in which they exchange arguments for or against correspondences. When they find a mutual agreement they can transform the resulting alignment to a program that translates the exchanged messages or to axioms which, once integrated in the agent knowledge, enable interpretation of messages (see Fig. 1.9).

1.6.2 *Matching Contexts in Ambient Computing*

In ambient computing, applications running on mobile or embedded devices take advantage of the surrounding environment for providing services to users. Naturally, this environment undergo changes, e.g., with regard to user locations, and

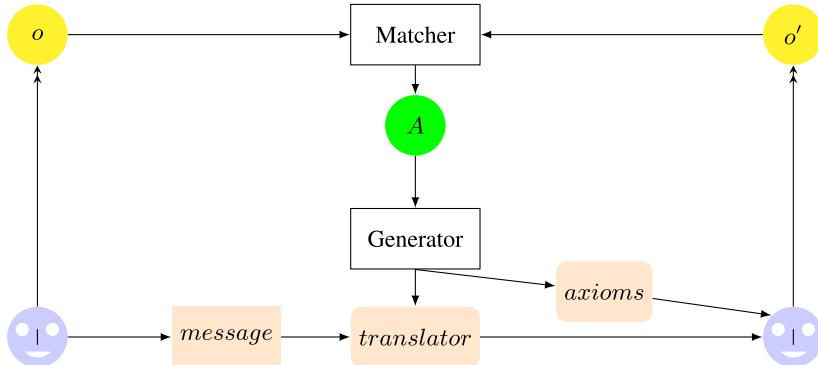


Fig. 1.9 Agent communication. In this scenario, it is useful to (1) *match* relevant parts of ontologies o and o' used by the agents, thus resulting in alignment A , (2) *generate* bridge *axioms* between two ontologies, and (3) incorporate the *axioms* into o' . Alternatively, the process can (2') *generate* a *message translator* from ontology o to ontology o' and (3') *apply* this *translator* to the *message*.

applications must always keep track of these changes, including newly appearing devices and sensors. Characterising the context in ambient computing goes through finding the information about the current situation in the environment by using various devices available in that environment, e.g., sensors. By doing so, applications provide context-aware solutions. If one wants to design flexible and smart ambient computing applications, it is useful to take advantage of the ontologies of these various devices, those of sensors available in the environments and their capabilities (Coutaz et al. 2005). Similar to web service descriptions, these ontologies will provide descriptions of the devices, even of abstract devices, such as a temperature service, and the way to interact with them.

Once again, it is expected that device providers will develop different ontologies adapted to their products or will extend some standard ontologies. Moreover, since applications evolve in ever changing environments in which devices can fail and new ones can appear, there is no way to freeze once and for all the ontologies that are relevant and available at a particular moment.

Therefore, in order to properly operate in ambient computing environments, applications have to be expressed in terms of generic features that are matched against the actual environment. This can take advantage of ontology matching (Euzenat et al. 2008b; Suarez Meza et al. 2011), since similar devices are likely to be used by similar applications. Thus, providing a service for reconciling various ontologies and storing the results obtained from previous interactions should help these applications in sharing and reusing the established alignments.

1.7 Navigation and Query Answering on the Web

This section presents several applications some of which extend the web experience to the semantic web by using resources such as formal ontologies. Operating

in an open environment, these applications most often require matching. In particular, the applications under consideration include navigation on the semantic web (Sect. 1.7.1), query answering on the web (Sect. 1.7.2), and query answering on the deep web (Sect. 1.7.3). Below we provide a short motivating scenario from the travelling domain, which is concerned with vertical search engines.

Motivating Scenario from the Travelling Domain A typical situation when one plans a business or a vacation trip is that of choosing a flight leading to a destination. This often happens through comparing fares, airports and times of the multiple competing alternatives. Thus, various flights are examined, a suitable one is selected, for instance, based on its price, and a ticket is purchased. The compared data comes from dynamic resources, whose contents are accessible through querying on-line databases, and cannot be accessed directly through static URL links. Traditional access to such (deep) web resources is performed by manually filling the multiple underlying query interfaces one by one. In turn, automatic access, provided by domain portals or vertical search engines, needs understanding of such interfaces and submission of valid values to the respective resources in order to offer a unified access and view on the search data, such as on airfares (Chang 2009). Matching is needed to identify the corresponding fields across relevant resources available on the web in order to enable appropriate query reformulation and answering with respect to each of the engaged resources.

1.7.1 *Navigation on the Semantics Web*

Browsers such as Magpie (Dzbor et al. 2003, 2004; Gridinoc et al. 2008) are designed to take advantage of semantic annotations associated with web pages. For instance, Magpie can recognise manifestations of instances of an ontology in a web page, display these instances specifically (different colours for different classes) and add services such as linking to the instance web page.

In open web browsing, the key point is to be able to select, at run time, the appropriate ontologies for the given browsing context. Indeed, web pages are linked to other web pages whose content may notably differ from that of the source page. In order to improve user experience, it is necessary to take new ontologies into account dynamically and to be able to connect them to the already available ontologies. Thus, ontology matching is needed to match between a set of terms that describes the topic of the current page and the relevant on-line ontologies.

For instance, the following short news story is about both trips to exotic locations and talks (Sabou et al. 2006):

For April and May 2005, adventurer Lorenzo Gariano was part of a ten-man collaborative expedition between 7summits.com and the 7summits club from Russia, led by Alex Abramov and Harry Kikstra, to the North Face of Everest. This evening he will present a talk on his experiences, together with some of the fantastic photos he took.

An ontology that covers such concepts as adventurer, expedition, talk and photos should be selected or discovered from the web. This requires that the above mentioned concepts are matched to the corresponding concepts from the available on-line ontologies. In addition, if some of the search terms cannot be found in an ontology, correspondences with more or less general concepts in the ontology are acceptable. Finally, not all the entities of the ontology need to be involved in matching. It is sufficient to consider only those entities that are similar to the terms found on the web page.

1.7.2 *Query Answering on the Web*

Contrary to the information integration scenario (Sect. 1.2), information on the web is not described by a global schema over which queries can be expressed. Moreover, users are used to query the web using their own terminology. Then, a semantic query answering system on the web has to rewrite the query with respect to available ontologies in order to use reasoning for providing answers.

For instance, a query answering system such as AquaLog (Lopez et al. 2005) is aware of an ontology about academic life which has been populated to describe knowledge related to some university (Sabou et al. 2006). For answering a query such as *Which projects are related to researchers working with ontologies?*, AquaLog interprets it in terms of entities available in the system ontology. For this, it first translates this query into the following triples: $\langle \text{projects}, \text{related to}, \text{researchers} \rangle$ and $\langle \text{researchers}, \text{working with}, \text{ontologies} \rangle$. Then it attempts to match these triples to the concepts of the underlying ontology. For example, the term projects should be identified to be equivalent to the ontology concept Project and ontologies is assumed equivalent to the ontologies instance of the Research-Area concept. If Action is a subclass of Project, the system will be able to take actions into account in its answers.

The scope of AquaLog is limited by the amount of knowledge encoded in the ontology of the system. This is similar to the efforts brought by search engines to promote one common schema: schema.org.¹⁰ A latter version of AquaLog, called PowerAqua (Lopez et al. 2006), extends its predecessor, as well as some other systems with similar goals, such as Observer (Mena et al. 1996), towards open query answering. PowerAqua aims at selecting and aggregating information derived from multiple heterogeneous ontologies on the web. Matching constitutes the core of this selection task. Unlike in AquaLog, matching is now performed between the triples and many on-line ontologies (not just the single ontology of the system). It is not necessary to match all query triples within one ontology. When no ontology concept is found for an element of a triple, the use of more general concepts is also acceptable. Moreover, it is not necessary to try to match the whole ontology against the query, but only the relevant fragments. Finally, an early version of the Aqua question answering system (Vargas-Vera and Motta 2004) was also coupled with the DSSim (Sect. 8.1.44) matching system.

¹⁰<http://schema.org>.

1.7.3 Query Answering on the Deep Web

The so-called *deep web* is made of the web sites searchable via query interfaces (HTML forms) giving access to one or more back-end databases. It is believed that it contains much more information (Chang et al. 2004; He et al. 2007; Chang 2009; Cafarella et al. 2011) than the billions of static HTML pages. At the moment, search engines are not very effective at crawling and indexing the deep web, since they cannot handle meaningfully the query interfaces.

Thus, users have difficulties, first in discovering the relevant deep web resources and then in querying them. A standard travelling scenario was described in the beginning of Sect. 1.7. Another example concerns buying a book with the lowest price among multiple on-line book stores. Query interfaces can be viewed as simple schemas (sets of terms). For example, in the book selling domain, the query interface of an on-line bookstore can be considered as a schema represented as a set of concept attributes, namely Author, Title, Subject, ISBN, Publisher. Thus, in order to enable query answering from multiple sources on the deep web, it is necessary to identify semantic correspondences between the attributes of the query interfaces of the web sites. This is akin to ontology matching. Ultimately, these correspondences are used for the on-the-fly translation of a query between interfaces of the web databases.

1.8 Summary

The above panorama shows a widespread need for ontology (in a wide sense) matching. Moreover, the need for matching is not limited to one particular application. In fact, it exists in any application that communicates through ontologies. Thus, it is natural that in the future more fine-grained or evolved examples of applications requiring matching will appear, e.g., ontology repair (McNeill 2006) and web APIs discovery (Taheriyan et al. 2012), or new ones, such as data interlinking on the web of data.

Since semantic heterogeneity is an intrinsic problem of any application involving more than one party, it is reasonable to consider ontology matching as a unified object of study. However, there are notable differences in the way these applications use matching. Application-related differences must be clearly identified in order to provide the best suited solution in each case.

Figure 1.10 orders these applications according to their *dynamics*, i.e., autonomy of parties participating in an application and rate of changes in an application. This shows that agent communication and query answering have a more dynamic profile compared to other applications. In fact, agents, besides having the ability to enter or leave the network or to change their ontologies at any moment (as in the peer-to-peer case), are also able to negotiate alignments and potential mismatches. Data integration, data interlinking, and merchant catalogue matching, due to the multiplicity of stakeholders, have a higher dynamics than ontology import and schema integration, where typically only a small and limited number of parties participate. Finally, the three bottom applications represent traditional applications, while the

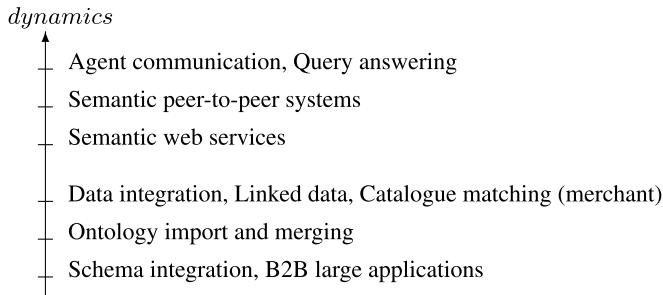


Fig. 1.10 Distribution of some applications with regard to their dynamics.

three top applications can be considered as dynamic applications. The uneven step in the middle of the dynamics axis in Fig. 1.10 is used to stress the above mentioned distinction.

Another dimension along which these applications differ is the purpose for which they perform matching:

- ontology engineering requires the ability to *transform* relevant ontologies or some parts of these ontologies into an ontology focussing on a domain of interest being modelled or to generate a set of bridge axioms that will help in identifying corresponding concepts (the transformations apply at the ontological level);
- schema integration requires the ability to *merge* schemas under consideration into a single schema (the transformations apply at the ontological level and instance translation applies at the data level);
- data integration requires the ability to *translate data* instances residing in multiple local schemas according to a global schema definition in order to enable query answering;
- data interlinking requires to *link* data instances on the web of data;
- peer-to-peer systems and more generally query answering systems require bidirectional *mediators* able to transform queries (ontological level) and translate back answers (data level);
- agent communication requires *translators* for messages sent by one agent to another, which apply at the data level; similarly, semantic web services require one-way data translations for composing services.

As this brief overview indicates, there are many different applications which require or can take advantage of matching ontologies. However, in spite of a common need for matching, application matching requirements are quite different. In particular, one can distinguish between traditional and dynamic applications both from the dynamics standpoint and the requirement standpoint. These two observations justify both the unified treatment of matching that we take in this book and the position of considering ontology matching being a separate operation, as opposed to considering merging or mediating being the primitive operations.

The next chapter will provide a more precise definition to this unified view of matching.

Chapter 2

The Matching Problem

In a distributed and open system, such as the semantic web and many of the applications presented in the previous chapter, heterogeneity cannot be avoided. Different actors have different interests and habits, use different tools and knowledge, and most often, at different levels of detail. These various reasons for heterogeneity lead to diverse forms of heterogeneity, and, therefore, should be carefully taken into consideration.

In this chapter, we first present existing ways of expressing knowledge that are found in diverse applications (Sect. 2.1). We then discuss in more detail ontologies and ontology languages as knowledge representation formalisms (Sect. 2.2). We introduce several justifications for heterogeneity (Sect. 2.3). These should help in designing a matching strategy with respect to the kind of heterogeneity that has to be faced. Then, we briefly review some terminology related to matching and alignment as well as provide the meaning that will be used for these terms in this book (Sect. 2.4). Finally, we give a formal account of the matching problem by offering a semantics for the matching result, i.e., the alignment (Sect. 2.5).

Our goal here is not to close the debate by providing some ultimate semantics for alignments or by settling the definitive meaning of terms, but rather to give definitions that help readers in understanding better the matching solutions that are presented in this book, as well as the results they produce.

2.1 Vocabularies, Schemas and Ontologies

So far we have considered ontologies without being precise about their meaning. An ontology can be viewed as a set of assertions that are meant to model some particular domain. Usually, they define a vocabulary used by a particular application. In various areas of computer science, there are different data and conceptual models that can be thought of as ontologies. These are, for instance, folksonomies, database schemas, UML models, directories, thesauri, XML schemas and formal ontologies (axiomatised theories). These and other examples are given in decreasing order of

formality in Fig. 2.1. Thus, a top-level ontology is supposed to have a well-defined explicit semantics, whereas the interpretation of directories in a file system is mostly implicit. In fact, it depends only on what its creator had in mind, i.e., the meaning of labels, the background knowledge, and the context in which these labels occur are all implicit, and therefore, these are not a part of the directory specification.

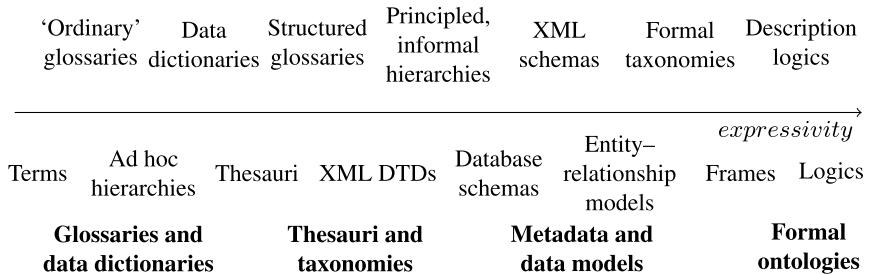


Fig. 2.1 Various forms of ontologies ordered by their expressivity (adapted from (Uschold and Gruninger 2004)).

We provide below examples of various forms of ontologies of Fig. 2.1 and illustrate some heterogeneity problems encountered in these forms.

2.1.1 Tags and Folksonomies

Tags and *folksonomies* are used as very simple ways to describe a corpus of knowledge by just giving names, called tags, to them. This is used in popular web sites, such as del.icio.us¹ for web site annotation, or Flickr² for annotating pictures. An example of tags for books and book collections is given in Fig. 2.2.

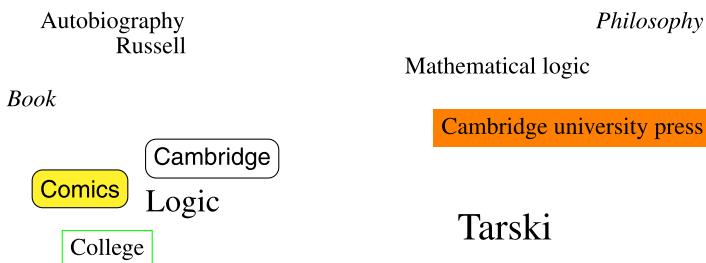


Fig. 2.2 Fragments of two folksonomies.

¹<http://del.icio.us>.

²<http://www.flickr.com>.

Obviously, different users use different tags. Even if these tags remain internally coherent for users who created them, this internal structure is not explicit for the machine. It is difficult to find relations between the tags of two folksonomies. Moreover, the fact that these tags do not have direct relations with each other (in one folksonomy) makes that problem even harder. However, there has been work aiming at inducing a structure between the tags, e.g., Flickr clusters. These are based mostly on the set of objects, e.g., pictures, web sites, that are indexed by the corresponding tags.

2.1.2 Directories

A taxonomy is a partially ordered set of taxons (classes) in which one taxon is greater than another one only if what the former denotes includes what is denoted by the latter. Directories or classifications are taxonomies that are used by companies for presenting goods on sale, by libraries for storing books, or by individuals to classify files on a personal computer. Some well-known examples of directories include that of Yahoo³ and the Open Directory Project.⁴ These directories are hierarchies of folders identified by labels and containing items, such as bookmarks, or goods. The semantics of these folders is given by the items they ultimately contain (Giunchiglia et al. 2006a). Of course, each independent entity tends to develop its own directory based on its own needs and tastes (see Fig. 2.3).

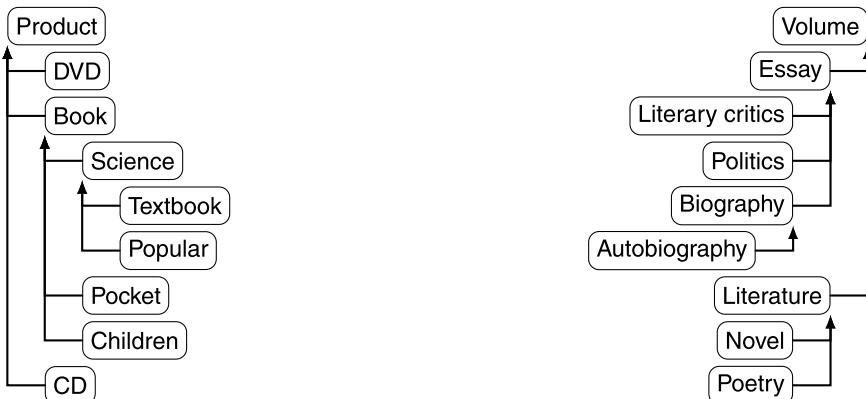


Fig. 2.3 Fragments of two directories.

In Fig. 2.3, the directory on the left represents the set of items of a bookstore or a cultural good seller, while the one on the right is the directory of a person that illustrates the content of his or her personal library. These directories encode

³<http://www.yahoo.com>.

⁴<http://dmoz.org>.

the domain under consideration at different levels of details, since these directories have been designed independently and for different purposes, i.e., selling versus classifying.

Finally, there exist some consensus classifications. In library science, the Dewey classification has been used for more than a century for classifying books by topics (Chan et al. 1996). In natural sciences, the principled classification of species represents another example (Schuh 1999). In medicine, UMLS⁵ (Unified Medical Language System) (Bodenreider 2004) is a unification of medical thesauri which is widely used.

2.1.3 Relational Database Schemas

Relational databases require data to be organised in a predefined way as tables or relations. A relational schema specifies the names of the tables as well as their types: the names and types of the columns of each table. The relational model also includes the notion of a key for each table: a subset of the columns that uniquely identifies each row (see Fig. 2.4). Finally, a column in a table may be specified as a foreign key pointing to a column in another table. This is used to keep referential constraints among various entities.

The schemas of Fig. 2.4 are presented with some data instances in tables. They display similar collections of information about books and authors, however, these are presented in different ways.

Relational databases, in a sense, are relatively restricted: table cells can only contain primitive data types, such as string or integer and cannot refer to some individual. For instance, the right-hand side schema of Fig. 2.4, in order to express the relationship between a book and its authors, requires an additional table expressing the authorship relation by joining the keys of both book and writer. Moreover, the relational model lacks the facility to organise data in a taxonomy. In both schemas of Fig. 2.4, tables corresponding to books have a type column assigning their class names to the objects. However, they indeed define a vocabulary used for expressing information. Several approaches have been proposed for overcoming this expressivity problem. For example, (i) by using a more expressive model, like the entity–relationship model (see Sect. 2.1.5) at design time and by generating a database out of it, or (ii) by using a more elaborate model, such as the object-oriented database model.

Widely used languages for specifying relational schemas, such as Structured Query Language (SQL) as well as some of its recent versions, e.g., SQL:1999 and SQL:2003, support many modelling capabilities, such as user-defined types, aggregation, generalisation, etc.

Finally, the use of a fixed schema tends to vanish with the current trend of dealing with large amount of data in a non-relational and flexible data model, often called

⁵<http://www.nlm.nih.gov/research/umls/>.

```

book (key: isbn):
    isbn -> int(11) auto_incr
    type -> varchar(10) [Volume]
    year -> int(11)
    title -> varchar(100)

item (key: id):
    id -> varchar(30)
    type -> varchar(10)
    price -> int(11) [NULL]
    name -> varchar(100)

creator (key: id, author):
    id -> varchar(30)
    author -> varchar(100)

```

id	type	price	name
89	Pocket	9.95	La chute
134	Popular	60	My life
77	Textbook		Introduction to logic
58	Science		Principia mathematica

id	name
89	Albert Camus
58	Alfred N. Whitehead
77	Alfred Tarski
134	Bertrand Russell
58	Bertrand Russell

isbn	type	year	title
2070360105	Novel	1956	La chute
0415189853	Autobiogr	1969	My life
048628462X	Essay	1941	Introduction to logic

firstname	middlename	lastname
Albert		Camus
Alfred		Whitehead
Alfred	North	Tarski
Bertrand		Russell

isbn	firstname	lastname
2070360105	Albert	Camus
2070394387	Albert	Camus
0521626064	Bertrand	Russell
0521626064	Alfred	Whitehead
0415189853	Bertrand	Russell
048628462X	Alfred	Tarski

Fig. 2.4 Fragments of two populated database schemas.

NoSQL (since data can be accessed through simpler operations than those in SQL). Schema matching can then be replaced by query matching if there are regularities in queries.

2.1.4 XML Schemas

Document Type Definitions (DTDs) and XML schemas have been introduced for specifying the structure of XML documents. The main ingredients of XML schemas include elements, attributes, and types. Elements can be either complex for specifying nested subelements, or simple for specifying built-in data types, such as string, for an element or attribute. XML schemas are rather complementary to directories: instead of describing how things are classified, they describe how things are made from the inside. For instance, the schema at the top of Fig. 2.5 describes the Product element that comprises a name element which is a string, an id which is a URI, a price which is a nonnegative integer, and topics which are strings. It also describes a Book element which is a Product that, in addition, has a sequence of authors which, in turn, are Person elements, and exactly one publisher. Even if element definitions can be extended or restricted as subcategories of a classification, the emphasis is on

```

<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="Person">
    <sequence><element name="name" type="xsd:string"/></sequence>
  </complexType>

  <simpleType name="creator"><restriction base="Person"/></simpleType>
  <simpleType name="author"><restriction base="creator"/></simpleType>

  <complexType name="Product">
    <sequence>
      <element ref="creator" minOccurs="1"/>
      <element name="name" type="xsd:string" minOccurs="1"/>
      <element name="id" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
      <element name="price" type="xsd:nonNegativeInteger" minOccurs="1"/>
      <element name="topic" type="xsd:string"/>
    </sequence>
  </complexType>

  <complexType name="Book">
    <complexContent>
      <extension base="Product">
        <sequence>
          <element ref="author" type="xsd:any"/>
          <element name="publisher" type="Publisher" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="Volume">
    <sequence>
      <element name="author" type="Writer" minOccurs="1"/>
      <element name="title" type="xsd:string" minOccurs="1"/>
      <element name="year" type="xsd:decimal"/>
    </sequence>
    <attribute name="isbn" type="xsd:anyURI"/>
  </complexType>

  <complexType name="Essay">
    <complexContent>
      <extension base="Volume">
        <sequence><element name="subject" type="xsd:any"/></sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="Human">
    <sequence>
      <element name="firstname" type="xsd:string"/>
      <element name="middlename" type="xsd:string"/>
      <element name="lastname" type="xsd:string"/>
    </sequence>
  </complexType>

  <complexType name="Writer">
    <complexContent><extension base="Human"/></complexContent>
  </complexType>
</schema>

```

Fig. 2.5 Fragments of two XML schemas.

their structure: the extension of an element is made by providing the elements which are modified in this structure. The sequential aspect of XML documents is part of the element specification, though it may be overruled.

In fact, these schemas are a shape according to which future documents are created, as opposed to an ontology, which is a description of existing, external objects. The specialisation hierarchy in XML schema is a type hierarchy that defines which kind of elements can occupy the place of another kind. For instance, if a shelf contains books, then putting a biography on this shelf is authorised. In principle, this classification structure does not have to correspond to any natural classification of the objects. Finally, it is worth noting that an XML-based description language, such as WSDL (Chinnici et al. 2007), is used to provide web service descriptions that often have to be matched in order to enable web service integration (Sect. 1.5).

2.1.5 Conceptual Models

Often, database researchers do not consider directly the relational schema but are rather concerned with the underlying entity–relationship model (Madhavan et al. 2002). Conceptual models cover what was properly described as such in (Brodie et al. 1984), as well as entity–relationship models (Chen 1976) that aim at abstracting databases, and UML (Booch et al. 1998) models that aim at abstracting object-oriented programs.

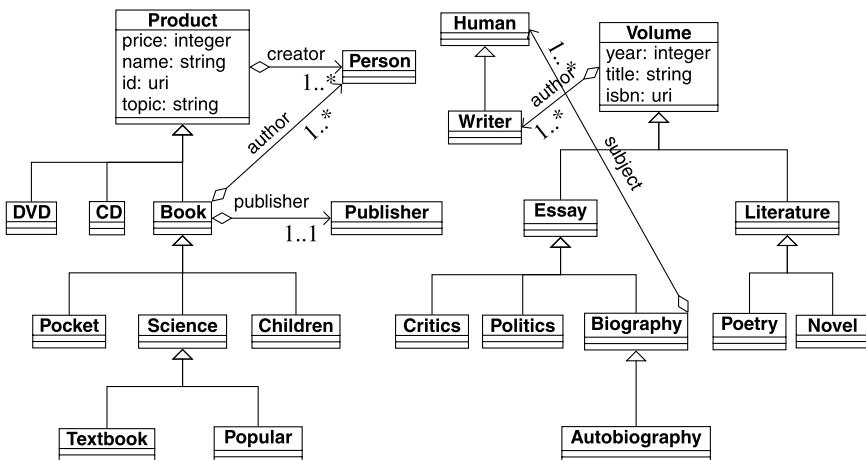


Fig. 2.6 Fragments of two conceptual models as UML class diagrams. Boxes describe entities and their internal structure; Specialisation is expressed by vertical triangular arrows; other relationships are displayed as regular arrows bearing a multiplicity indication.

These models offer a rich way of expressing entities which in this case can be meant as entities of some modelled domain, like people in a database, or specifica-

tion of entities to be created, like programs. They offer constructors for organising classes in a hierarchy as well as constructors for describing the internal structure of objects. They thus offer the best of both worlds: directories and databases. For instance, Fig. 2.6 describes two UML class diagrams corresponding to the same sort of models as presented before: a taxonomy of classes from an e-commerce site selling cultural goods on the left and a book library on the right. They both offer a complete description of the items through the specification of their properties and a taxonomy of classes. Moreover, they can express relationships between classes, e.g., that the author of a Book is a Person in the model on the left. The two models of Fig. 2.6 express comparable domains, e.g., a Volume will correspond to a Book, and yet largely different, e.g., there is no Product superclass in the right-hand side model.

2.1.6 *Ontologies*

It is nowadays common to see directories or conceptual models promoted as ontologies. Ontologies contain most of the features of entity–relationship models, and thus, most parts of the kinds of schemas considered above. The ontologies of Fig. 2.7 syntactically correspond to the models of Fig. 2.6.

Various types of ontologies can be distinguished: foundational ontologies are those providing an axiomatisation of other ontologies in terms of fundamental concepts (perdurant, endurant, etc.). They are very often complemented by an upper-level ontology, which defines commonplace non-foundational concepts (vehicles, people, etc.). They may be seen as horizontal ontologies. Examples are the upper Cyc ontology (Lenat and Guha 1990), the Suggested Upper Merged Ontology (SUMO) (Niles and Pease 2001), the Unified Foundational Ontology (UFO) (Guizzardi 2005), and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) (Gangemi et al. 2003).

Domain ontologies are vertical ontologies that consider concepts relevant to a particular domain such as cooking or biology. This is typically the case of the Foundational Model of Anatomy⁶ (FMA) developed at the University of Washington in the medical field of anatomy. Finally, application-specific ontologies contain the concepts specific to a particular application, e.g., the gene ontology for genes.

The distinctive feature of ontologies is the existence of a model-theoretic semantics: ontologies are logic theories. Ontology interpretation is not left to users who read the diagrams or to knowledge management systems implementing them, it is specified explicitly. The semantics provides the rules for interpreting the syntax that do not provide the meaning directly, but constrains the possible interpretations of what is declared.

It is commonplace in theoretical database research to consider relational databases with a first-order semantics. However, this is not part of the official SQL

⁶<http://sig.biostr.washington.edu/projects/fm/>.

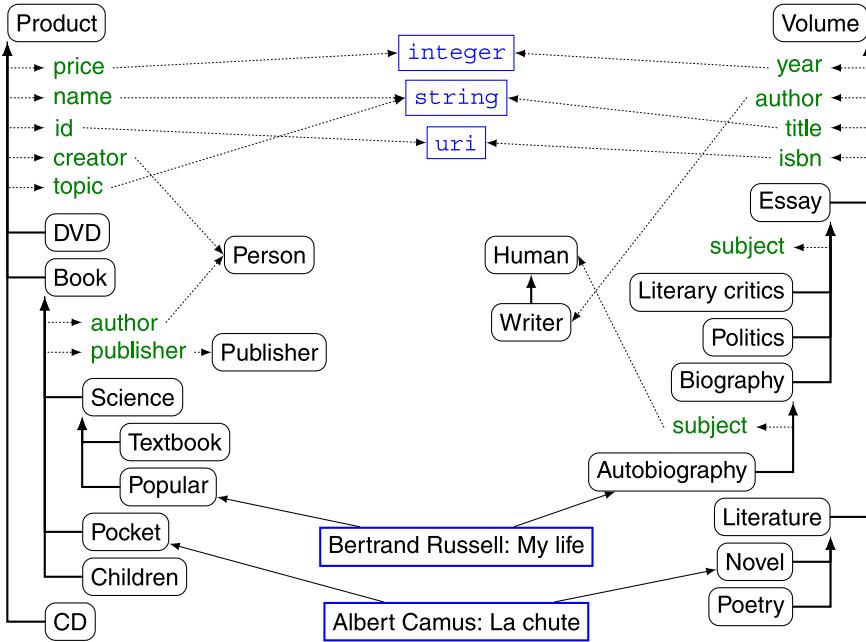


Fig. 2.7 Fragments of two ontologies.

standard (Melton 2003). Moreover, the relational algebra used in database schemas is not very expressive: expressiveness resides in the query language.

For these reasons of rich expressiveness and presence of a model-theoretic semantics, we will specifically focus on ontologies. Traditionally, ontologies were considered different from knowledge bases, like a database schema is different from a database that uses it. We will not enforce this distinction here and only use the term ‘ontology’. We thus discuss ontologies in more detail with the idea that these discussions are for part relevant to other kinds of conceptual models.

The semantics of ontologies may be constrained by additional axioms. This could be, in some languages, the opportunity to add statements, such as an autobiography is a biography whose topic is the author:

$$\forall x, \quad \text{Autobiography}(x) \Rightarrow \exists y; \quad \text{Person}(y) \wedge \text{author}(x, y) \wedge \text{topic}(x, y)$$

For the sake of completeness, we give in the next section a syntax and semantics for a minimal ontology language.

2.2 Ontology Language

Ontologies are expressed in an ontology language. There are a large variety of languages for expressing ontologies (Staab and Studer 2004). Fortunately, most of these

languages share the same kinds of entities, often with different names but comparable interpretations. We briefly describe which entities are found in ontology languages. It is not our purpose to commit to one particular language, however this section aims to facilitate the understanding of some of the forthcoming examples which are given in OWL (Motik et al. 2009a, 2009b), an ontology language recommended by the W3C.

2.2.1 Ontology Entities

Ontology languages usually deal with the following kinds of entities:

Classes or concepts are the main entities of an ontology. These are interpreted as a set of individuals in the domain. They are introduced in OWL by the `owl:Class` construct. For example, in Fig. 2.7, Book and Person are classes.

Individuals or objects or instances are interpreted as particular individual of a domain. These are introduced in OWL by the `owl:Thing` construct. For example, in Fig. 2.7, the objects Albert Camus: La chute and Bertrand Russell: My life are individuals.

Relations are the ideal notion of a relation independently to what it applies. Relations are interpreted as a subset of the product of the domain. These are introduced in OWL by the `owl:ObjectProperty` or `owl:DatatypeProperty` construct. For example, in Fig. 2.7, creator and topic are relations.

Data types are particular parts of the domain which specify values. Contrary to individuals, values do not have identities. For example, in Fig. 2.7, String and Integer are data types.

Data values are simple values. For example, in Fig. 2.7, the string ‘My life’ is a data value that can be the title of an Autobiography.

These entities do not have to be named. They may be constructed out of other entities. In OWL, a concept can be created out of the restriction of a relation. For example, this occurs if one defines the class Writer as the set of individuals that have written something:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasWritten" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
</owl:Restriction>
```

Alternatively, a new class can also be constructed by combining two other classes. For example, when considering that a low price pocket book for children is a Pocket book that is also a Children and LowPrice book:

```
<owl:intersectionOf>
  <owl:Class rdf:resource="#Pocket" />
  <owl:Class rdf:resource="#Children" />
  <owl:Class rdf:resource="#LowPrice" />
</owl:intersectionOf>
```

Entities may be connected by various kinds of relations, including:

Specialisation, or subsumption, between two classes or two properties is interpreted as inclusion of the interpretations. For instance, in Fig. 2.7, the class Book is a specialisation of the class Product. Specialisation is introduced in OWL by the `rdfs:subClassOf` or `rdfs:subPropertyOf` constructs.

Exclusion, or disjointness, between two classes or two properties is interpreted as the emptiness of the intersection of their interpretations. For instance, in Fig. 2.7, the class Product could be declared to be exclusive to the class Person. Exclusion is introduced in OWL by the `owl:disjointWith` construct.

Instantiation, or typing, between individuals and classes, property instances and properties, values and data types is interpreted as membership. For instance, in Fig. 2.7, the product presented as Bertrand Russell: My life is an instance of the class Popular. Instantiation is expressed in OWL with the `rdf:type` construct.

Example 2.1 The class Book of Fig. 2.7 can be expressed in OWL as follows:

```
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf rdf:resource="#Product" />
  <rdfs:label xml:lang="en">book</rdfs:label>
  <rdfs:comment xml:lang="en">A book.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author" />
      <owl:cardinality rdf:datatype="#xsd:nonNegativeInteger">
        1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#publisher" />
      <owl:allValuesFrom rdf:resource="#Publisher" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

In particular, this defines two classes by restricting the cardinality of the author relation and restricting the range of the publisher relation. It also relates the defined Book class to these classes and the Product class by specialisation (`rdfs:subClassOf`).

In summary, we can assume an ontology to be characterised as follows.

Definition 2.2 (Ontology) An ontology is a tuple $\mathcal{O} = \langle C, I, R, T, V, \sqsubseteq, \perp, \in, = \rangle$, such that

C is the set of classes;

I is the set of individuals;

R is the set of relations;

T is the set of data types;

V is the set of values (C, I, R, T, V being pairwise disjoint);

\sqsubseteq is a relation on $(C \times C) \cup (R \times R) \cup (T \times T)$ called specialisation;

\perp is a relation on $(C \times C) \cup (R \times R) \cup (T \times T)$ called exclusion;

\in is a relation over $(I \times C) \cup (V \times T)$ called instantiation;

$=$ is a relation over $I \times R \times (I \cup V)$ called assignment.

Ontology formulas may contain more than these assertions, e.g., quantified assertions or assertions related by logical connectors. We will restrict ourselves to relations between entities.

Many algorithms transform these ontologies into labelled graphs, where nodes are typed. We use such a notation in the diagrams of this book (see Fig. A.1 of Appendix A for details).

2.2.2 Ontology Language Semantics

The semantics of ontology languages is usually given through model theory. In particular, it defines an interpretation function that maps each ontology entity to a set D called the domain of interpretation.

Definition 2.3 (Interpretation) Given an ontology $\mathcal{O} = \langle C, I, R, T, V, \sqsubseteq, \perp, \in, = \rangle$, an interpretation of \mathcal{O} is a pair $\langle I, D \rangle$, such that D is called the domain of interpretation and I is a function called the interpretation function, such that

- $\forall c \in C, I(c) \subseteq D$,
- $\forall r \in R, I(r) \subseteq D \times (D \cup V)$,
- $\forall i \in I, I(i) \in D$,
- $\forall t \in T, I(t) \subseteq V$,
- $\forall v \in V, I(v) \in V$.

An assertion expressed in an ontology language is said to be satisfied by an interpretation if the interpretation is coherent with this assertion.

Definition 2.4 (Satisfaction of a formula) Given an ontology $\mathcal{O} = \langle C, I, R, T, V, \sqsubseteq, \perp, \in, = \rangle$, and a formula δ , the satisfaction of δ by an interpretation $\langle I, D \rangle$ of \mathcal{O} (denoted by $I \models \delta$), is defined as follows:

$I \models c \sqsubseteq c'$	if and only if	$I(c) \subseteq I(c')$
$I \models r \sqsubseteq r'$	if and only if	$I(r) \subseteq I(r')$
$I \models t \sqsubseteq t'$	if and only if	$I(t) \subseteq I(t')$
$I \models c \perp c'$	if and only if	$I(c) \cap I(c') = \emptyset$
$I \models r \perp r'$	if and only if	$I(r) \cap I(r') = \emptyset$
$I \models t \perp t'$	if and only if	$I(t) \cap I(t') = \emptyset$
$I \models i \in c$	if and only if	$I(i) \in I(c)$
$I \models v \in t$	if and only if	$I(v) \in I(t)$
$I \models i.r = i'$	if and only if	$\langle I(i), I(i') \rangle \in I(r)$
$I \models i.r = v$	if and only if	$\langle I(i), I(v) \rangle \in I(r)$

So, an ontology is a set of assertions that selects the set of interpretations that satisfies them. These interpretations are called models. They constitute the possible interpretations of an ontology.

Definition 2.5 (Model) Given an ontology \mathcal{O} , a model of \mathcal{O} is an interpretation $m = \langle I, D \rangle$ of \mathcal{O} , which satisfies all the assertions in \mathcal{O} :

$$\forall \delta \in \mathcal{O}, \quad m \models \delta$$

The set of models of an ontology \mathcal{O} is denoted by $\mathcal{M}(\mathcal{O})$.

Finally, an important notion is the set of assertions that are consequences of an ontology. These are the assertions implicitly entailed by an ontology and they determine the answers to queries.

Definition 2.6 (Consequence) Given an ontology formula δ , δ is a consequence of an ontology \mathcal{O} , if and only if, it is satisfied by all models of \mathcal{O} . This is denoted by $\mathcal{O} \models \delta$.

Given a model m , we will denote by $m(e)$ the application of the interpretation function of the model to some ontology entity e .

This digression introduced more precisely, albeit generally, a simplified syntax and semantics for ontologies. This will be useful when considering the meaning of matching ontologies.

2.3 Types of Heterogeneity

The goal of matching ontologies is to reduce heterogeneity between them. Heterogeneity does not lie solely in the differences between goals of the applications according to which they have been designed or in the expression formalisms in which ontologies have been encoded. There have been many different classifications to types of heterogeneity (Batini et al. 1986; Sheth and Larson 1990; Breitbart 1990; Kim and Seo 1991; Goh 1997; Hull 1997; Kashyap and Sheth 1998; Benerecetti et al. 2000; Wache et al. 2001; Klein 2001; Euzenat 2001; Corcho 2005; Hameed et al. 2004; Ghidini and Giunchiglia 2004; Bouquet et al. 2004a). Some of them focus on mismatches (Klein 2001), others rather mention interoperability levels (Euzenat 2001). We consider here the most obvious types of heterogeneity:

Syntactic heterogeneity occurs when two ontologies are not expressed in the same ontology language. This obviously happens when comparing, for instance, a directory with a conceptual model. This also happens when two ontologies are modelled by using different knowledge representation formalisms, for instance, OWL and F-logic. This kind of mismatch is generally tackled at the theoretical level when one establishes equivalences between constructs of different languages. Thus, it is sometimes possible to translate ontologies between different

ontology languages whilst still preserving their meaning (Euzenat and Stuckenschmidt 2003).

Terminological heterogeneity occurs due to variations in names when referring to the same entities in different ontologies. This may be caused by the use of different natural languages, e.g., Paper vs. Articulo, different technical sublanguages, e.g., Paper vs. Memo, or the use of synonyms, e.g., Paper vs. Article.

Conceptual heterogeneity, also called semantic heterogeneity in (Euzenat 2001) and logical mismatch in (Klein 2001), stands for the differences in modelling the same domain of interest. This may happen due to the use of different (and, sometimes, equivalent) axioms for defining concepts or due to the use of totally different concepts, e.g., geometry axiomatised with points as primitive objects or geometry axiomatised with spheres as primitive objects. As noted in (Klein 2001) and (Visser et al. 1998), there is a difference between the conceptualisation mismatch, which relies on the differences between modelled concepts, and the explication mismatch, which relies on the way these concepts are expressed. (Visser et al. 1998) provides a precise classification of these mismatches.

Finally, in the context of conceptual differences, (Benerecetti et al. 2001) identifies three important reasons for these to hold. We discuss these below and give examples with the help of the notion of a geographic map:

- *Difference in coverage* occurs when two ontologies describe different, possibly overlapping, domains at the same level of detail and from a unique perspective. This is obviously the case for two partially overlapping geographic maps.
- *Difference in granularity* occurs when two ontologies describe the same domain from the same perspective but at different levels of detail. This applies to geographic maps with different scales, e.g., one displays buildings, while another depicts whole cities as points.
- *Difference in perspective*, also called difference in scope (Chalupsky 2000), occurs when two ontologies describe the same domain, at the same level of detail, but from a different perspective. This occurs for maps with different purposes: a political map and a geological map do not display the same objects.

Semiotic heterogeneity, also called pragmatic heterogeneity in (Bouquet et al. 2004a), is concerned with how entities are interpreted by people. Indeed, entities which have exactly the same semantic interpretation are often interpreted by humans with regard to the context, for instance, of how they are ultimately used. This kind of heterogeneity is difficult for the computer to detect and even more difficult to solve, because it is out of its reach. The intended use of entities has a great impact on their interpretation, therefore, matching entities which are not meant to be used in the same context is often error-prone. Given the limited grasp that a computer can have on these issues, we do not deal with semiotic heterogeneity here.

Usually, several types of heterogeneity occur together. This book is only concerned with reducing the terminological and conceptual types of heterogeneity. Techniques for dealing with these types individually are presented in Chap. 5, while techniques for considering them together are provided in Chaps. 6 and 7.

2.4 Terminology

As can be observed from what we have presented so far, in the area of ontology matching, different authors including ourselves use different terms to refer to similar concepts, and, vice versa, sometimes different concepts are referred to by the same term (Chalupsky 2000; Klein 2001; Euzenat 2001; Noy and Klein 2004; Kalfoglou and Schorlemmer 2003b; Bouquet et al. 2004a; Bellahsene et al. 2011). This is especially confusing since these terms can be used for describing both an action and its result, e.g., mapping. In this section, we provide a working glossary with the definitions of terms as they are going to be used in this book. They are stable and have not changed since its first edition, though several new entries have been added, such as networks of ontologies and data interlinking.

Matching is the process of finding relationships or correspondences between entities of different ontologies.

Alignment is a set of correspondences between two or more (in case of multiple matching) ontologies (by analogy with molecular sequence alignment). The alignment is the output of the matching process.

Correspondence is the relation holding, or supposed to hold according to a particular alignment, between entities of different ontologies. These entities can be as different as classes, individuals, properties or formulas. Some authors use the term mapping instead, however, it will not be used in this sense in this book.

Anchor is a correspondence between two ontology entities, which are usually required prior to performing other tasks, such as reasoning, partitioning and also matching.

Mapping is the oriented, or directed, version of an alignment: it maps the entities of one ontology to at most one entity of another ontology. This complies with the mathematical definition of a mapping instead of that of a general relation. The mathematical definition would in principle require that the mapped object is equal to its image, i.e., that the relation is an equivalence relation. A mapping can be seen as a collection of mapping rules all oriented in the same direction, i.e., from one ontology to the other, and such that the elements of the source ontology appear at most once.

Mapping rule is a correspondence which maps an entity of one ontology into another one from another ontology.

Ontology merging is the creation of a new ontology from two, possibly overlapping, source ontologies. The initial ontologies remain unaltered. The merged ontology is assumed to contain the knowledge of the initial ontologies, e.g., consequences of each ontology are consequences of the merge. This concept is closely related to that of schema integration in databases.

Ontology integration is the inclusion in one ontology of another ontology and assertions expressing the *glue* between these ontologies, usually as bridge axioms. The integrated ontology is assumed to contain the knowledge of both initial ontologies. Contrary to merging, the first ontology is unaltered while the second one is modified.

Bridge axioms or articulation axioms are formulas, in an ontology language, that express the alignments such that it is possible to integrate the entities of an ontology within one another. Bridge axioms are the basis for ontology merging when the ontologies are expressed in the same language.

Ontology translation is the process of transforming an ontology from one ontology language to another. By extension, it is a program for translating ontologies.

Ontology transformation is the process of expressing the entities of an ontology with respect to the entities of another ontology, i.e., relations between entities of the first ontology and those of the second one are added to the first ontology. So the initial consequences of the first ontology are still consequences of the transformation result. The two initial ontologies are unaltered and a third ontology, the transformation result, is created. By extension, it is a program that transforms ontologies.

Data translation is the process of transforming data or instances from one ontology into corresponding data or instances expressed with respect to another ontology. By extension, it is a program that translates data.

Data interlinking is the process of establishing explicit links, e.g., equivalences, between instances from different data sources.

Mediation consists of interfacing two software components by dynamically altering the information stream between these. By extension, a mediator is a program performing mediation. In web service composition, a mediator translates the output of a service into the input of another one: it thus performs data translation. In query answering applications it is a dual pair of translations that transforms the query from one ontology to another and that translates the answers back.

Networks of ontologies are made of a set of ontologies and a set of alignments between these ontologies. The ontologies may be related for several reasons: they may be complementary; they may be two independent domain ontologies, e.g., sales and tyres, refinement; there may be a domain ontology specialising a top-level ontology; or they may be supplementary, e.g., a version replacing another version or two ontologies about the same domain.

Linked data, also known as *web of data*, is a global data space, which is made of structured data sources published on the web, e.g., in RDF, that are explicitly interlinked.

Ontology version of an ontology is the ontology resulting from the application of modifications to this ontology.

Ontology reconciliation is a process that harmonises the content of two or more ontologies, typically requiring changes on one of the two sides or even on both sides (Hameed et al. 2004). In this case, there is no merging of the ontologies but co-evolution. Ontology reconciliation can be performed for the purpose of merging two ontologies or for the purpose of making them independent.

2.5 The Ontology Matching Problem

There have been different formalisations of matching and its result (Bernstein et al. 2000; Lenzerini 2002; Kalfoglou and Schorlemmer 2003b; Bouquet et al. 2004a; Zimmermann et al. 2006; Bellahsene et al. 2011). We provide here a general definition. It does not pretend to solve each particular problem nor to strictly cover the complete field. It aims at serving as a guide for this book.

2.5.1 The Matching Process

The *matching* operation determines an alignment A' for a pair of ontologies o and o' . There are some other parameters that can extend the definition of the matching process, namely (i) the use of an input alignment A , as known as anchor alignment, which is to be extended or completed by the process, (ii) the matching parameters, p , e.g., weights, thresholds, and (iii) external resources used by the matching process, r , e.g., common knowledge and domain-specific thesauri.

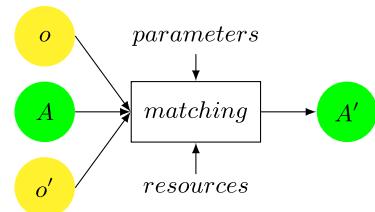
Technically, this process can be defined as follows:

Definition 2.7 (Matching process) The matching process can be seen as a function f which, from a pair of ontologies to match o and o' , an input alignment A , a set of parameters p and a set of oracles and resources r , returns an alignment A' between these ontologies:

$$A' = f(o, o', A, p, r)$$

This can be schematically represented as illustrated in Fig. 2.8.

Fig. 2.8 The matching process.



We use interchangeably the terms ‘matching operation’ (focussing on the input and the result), ‘matching task’ (focussing on the goal and the insertion of the task in a wider context) and ‘matching process’ (focussing on the internal processing).

It can be useful to specifically consider matching more than two ontologies within the same process. We call this multiple matching.

Definition 2.8 (Multiple matching process) The multiple matching process can be seen as a function f which, from a set of ontologies to match $\{o_1, \dots, o_n\}$, an input

alignment A , a set of parameters p and a set of oracles and resources r , returns an alignment A' between these ontologies:

$$A' = f(o_1, \dots, o_n, A, p, r)$$

The matching process is the main subject of this book. However, before discussing its internals, let us first consider what it provides: the alignment.

2.5.2 Structure of an Alignment

Alignments express correspondences between entities belonging to different ontologies. All definitions here are given for matching between two ontologies. In case of multiple matching, the definitions can be straightforwardly extended by using n -ary correspondences. A correspondence must consider the two corresponding entities and the relation that is supposed to hold between them. We provide a definition of alignments extending (Euzenat 2004; Bouquet et al. 2004a).

Since the related entities are an important part of alignments, they have to be defined. We separate the matched entities from the ontology language because it can be desirable to have a different language for identifying the matched entities. Given an ontology language, we use an *entity language* for expressing those entities that will be put in correspondence by matching. The expressions of this language depend on the ontology on which expressions are defined.

Definition 2.9 (Entity language) Given an ontology language L , an entity language Q_L is a function from any ontology $o \subseteq L$ which defines the matchable entities of ontology o .

The entity language can be simply made of all the formulas of the ontology language based on the ontology vocabulary. It can restrict its scope to particular kinds of formulas from the language, for instance, atomic formulas, or even to terms of the language, like class expressions. It can also restrict the entities to be only named entities. This is convenient in the context of the semantic web to restrict entities to those identifiable by their URIs. The entity language can also be an extension of the ontology language: this can be a query language, such as SPARQL (Prud'hommeaux and Seaborne 2008), adding operations for manipulating ontology entities that are not available in the ontology language itself, like concatenating strings or joining relations. Finally, the entity language may combine both extension and restriction, e.g., by authorising any Boolean operations over named ontology entities.

In the following we assume that each ontology interpretation can be extended to an interpretation of the entity language associated with the ontology.

The next important component of the alignment is the relation that holds between the entities. We identify a set of relations Θ that is used for expressing the

relations between entities. Matching algorithms primarily use the equivalence relation ($=$), meaning that the matched objects are the same or are equivalent if these are formulas. It is possible to use relations from the ontology language within Θ . For instance, using OWL, it is possible to take advantage of the `owl:equivalentClass`, `owl:disjointWith` or `rdfs:subClassOf` relations in order to relate classes of two ontologies. These relations correspond to set-theoretic relations between classes: *equivalence* ($=$); *disjointness* (\perp); *less general* (\leq). They can be used without reference to any ontology language. Finally, relations can be of any type and are not restricted to relations present within the ontology language, such as fuzzy relations or probability distributions over a complete set of relations, or similarity measures. Algebras of relations provide an alternative generalisation of these ad hoc sets of relations (Euzenat 2008).

With these ingredients, it is possible to define the correspondences that have to be found by matching algorithms.

Definition 2.10 (Correspondence) Given two ontologies o and o' with associated entity languages Q_L and $Q_{L'}$ and a set of alignment relations Θ , a *correspondence* is a triple:

$$\langle e, e', r \rangle,$$

such that

- $e \in Q_L(o)$ and $e' \in Q_{L'}(o')$;
- $r \in \Theta$.

The correspondence $\langle e, e', r \rangle$ asserts that the relation r holds between the ontology entities e and e' .

In the first edition of this book, this definition (numbered Definition 2.11) had two additional components: an identifier for each correspondence and a confidence measure. We now consider it better to keep correspondences as simplest expressions in order to facilitate the presentation. Identifiers and confidence measures, which are useful in practice, may be added to correspondences (see Example 2.11). In particular, we consider confidence measures as metadata (see Sect. 3.8).

Example 2.11 (Correspondence) For example, a simple kind of correspondence is as follows:

<http://book.ontologymatching.org/example/culture-shop.owl#Book> =
<http://book.ontologymatching.org/example/library.owl#Volume>

It asserts the equivalence relation between what is denoted by two URIs, namely the `Book` class in one ontology and the `Volume` class in another one. Some examples of more complex correspondences are as follows:

$\text{author}(x, concat(w.\text{firstname}, w.\text{lastname})) \Leftarrow .85$	$\wedge \text{writtenBy}(x, w)$ $\wedge \text{Writer}(w),$
--	---

which is a Horn clause expressing that, if there exists a Book x written by Writer w , the author of x in the first ontology is identified by the concatenation of the first and last name of w . This clause is additionally assigned a confidence degree of .85. Such a statement is also known as a full tuple-generating dependency.

Correspondences may also be expressed between queries as follows:

<pre>SELECT ?x ?y WHERE { ?x rdf:type o':Book . ?x o':author ?w . ?w o':name ?y . ?x o':topic ?y . }</pre>	$=$ <pre>SELECT ?x ?y AS (?u+ " "+?v) WHERE { ?x rdf:type o:Autobiography . ?x o:author ?z . ?z o:firstname ?u . ?z o:lastname ?v . }</pre>
--	--

which matches two queries returning a resource and a string representing pairs of books and person names, such that the book is the autobiography of the person.

There can be several possible correspondences for the same entities depending on the language in which correspondences are expressed. For instance, one could have the simple correspondence that speed in one ontology is equivalent to velocity in another one:

$$\text{speed} = \text{velocity}$$

or record that they are expressed in miles per hour and metre per second, respectively:

$$\text{speed} = \text{velocity} \times 2.237$$

$$0.447 \times \text{speed} = \text{velocity}$$

For pragmatic reasons, the relationship between two entities is often assigned a degree of confidence which can be viewed as a measure of trust in the fact that the correspondence holds—‘I trust 70 % the fact that the correspondence is correct or reliable’—and can be compared with the certainty measures provided by meteorological forecasts. This measure is taken from a confidence structure.

Definition 2.12 (Confidence structure) A confidence structure is an ordered set of degrees $\langle \mathcal{E}, \leq \rangle$ for which there exists the greatest element \top and the smallest element \perp .

The usage of confidence degrees is such that the higher the degree with regard to \leq , the more likely the relation holds. It is convenient to interpret the greatest element as the Boolean true and the least element as the Boolean false. In the following, we use the function $\kappa_A(\mu) : Q_L(o) \times Q'_L(o') \times \Theta \rightarrow \mathcal{E}$ to provide the confidence for a correspondence μ in an alignment A .

The most widely used structure is based on the real number unit interval $[0, 1]$, but some systems simply use the Boolean lattice. The lattice structure provides a way to aggregate measures (there always exist greatest lower bound and smallest upper bound to two confidences) and the stronger total order structure guarantees that it is always possible to compare two confidences. This is useful in evaluation (see Chap. 9). Some other possible structures are fuzzy degrees (Gal 2011), probabilities (Atencia et al. 2012a) or other lattices.

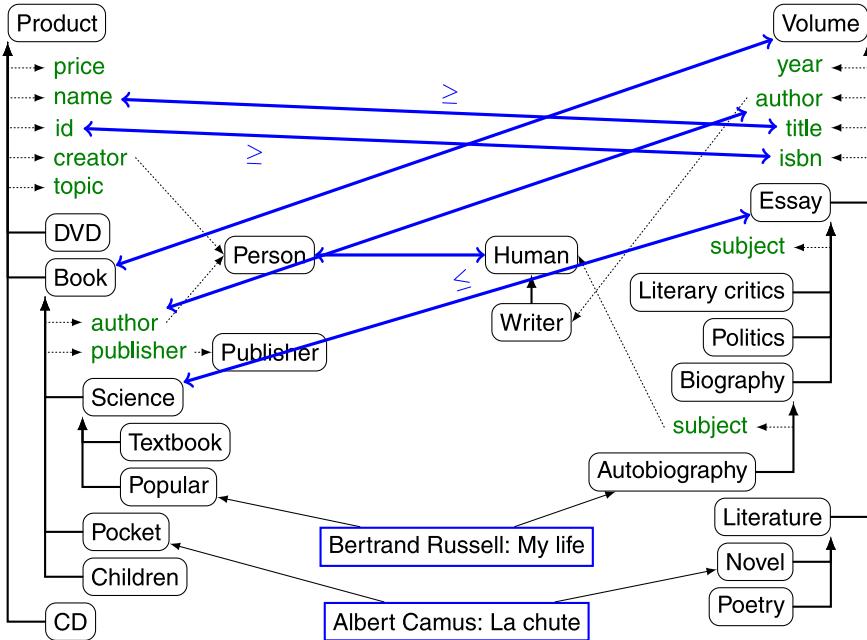


Fig. 2.9 Alignment between the ontologies of Fig. 2.7. Correspondences are shown as thick arrows that connect an entity from one ontology with an entity from another ontology. They are annotated with the relation that is expressed by the correspondence. By default the relation is $=$; otherwise, these are mentioned near the arrows.

Finally, an alignment is defined as a set of correspondences.

Definition 2.13 (Alignment) Given two ontologies o and o' , an *alignment* is a set of correspondences between pairs of entities belonging to $Q_L(o)$ and $Q_{L'}(o')$ respectively.

Example 2.14 (Alignment) Figure 2.9 displays a possible alignment for the pair of ontologies of Fig. 2.7. It can be expressed by the following correspondences:

$$\begin{array}{ll}
 \text{Book} = \text{Volume} & \text{name} \geq \text{title} \\
 \text{id} \geq \text{isbn} & \text{author} = \text{author} \\
 \text{Person} = \text{Human} & \text{Science} \leq \text{Essay}
 \end{array}$$

These definitions can be generalised to an arbitrary number of alignments and ontologies captured in the concept of a network of ontologies (or distributed system in the sense of (Ghidini and Serafini 1998; Franconi et al. 2003)), i.e., sets of ontologies and alignments.

Definition 2.15 (Network of ontologies) A network of ontologies $\langle \Omega, \Lambda \rangle$ is made of a set Ω of ontologies and a set Λ of alignments between these ontologies. We denote by $\Lambda(o, o')$ the set of alignments in Λ between o and o' .

The definition of alignments as sets of pairs of entities from two ontologies may seem simplistic. In fact it is very general and covers most of the cases encountered in the literature. Indeed, there are at least three types of $n:m$, multiple or complex alignments:

1. alignments involving more than two ontologies produced by multiple matching, that we may call multialignments,
2. alignments involving correspondences between more than two entities (still belonging to two ontologies),
3. alignments with entities involved in more than one correspondence that are denoted by the use of * (zero-or-more) or + (more-than-zero) in their cardinalities.

In case of multiple matching (1), the alignments must contain correspondences relating more than two entities. The definitions above must then be extended accordingly. This is not covered further here.

The second kind of correspondences (2) can be thought of as using nonbinary relations. For instance, in schema matching, some authors (Sheth and Larson 1990; Rahm and Bernstein 2001) tend to consider that a correspondence like

$$\text{address} = \text{street} + " " + \text{number}$$

is a ternary complex relation ($\cdot = \cdot + " " + \cdot$) between three entities address, street and number. However, given the nature of the ontology matching problem, we will consider that these objects can be grouped by operators in the entity language Q_L . These may include operators, such as concatenation, arithmetic operations or logical connectors, for that purpose. Hence, in our setting, the correspondence above is a simple correspondence in which the binary relation is equivalence (=) and the ontology entities are address and street + “ ” + number. This is the main reason why we consider ontology entities: the latter entity is a term built on strings and operations on strings (here concatenation, +). In its simplest expression, the only construction can be a set.

Option (3) is related to the multiplicity of the alignment if it is considered as a relation. By analogy with mathematical functions, it is useful to define some properties of the alignments. These apply when the only considered relation is equivalence (=) and confidence measures are not taken into account. One may ask for a total alignment with regard to one ontology, i.e., each of the entities of one ontology must be successfully mapped to another one. This property is purposeful whenever thoroughly transcribing knowledge from one ontology to another is the goal: there is no entity that cannot be translated.

One may also require the alignment to be injective with regard to one ontology, i.e., all the entities of the other ontology is part of at most one correspondence. In-

jectivity is useful in ensuring that entities that are distinct in one ontology remain distinct in the other one. In particular, this contributes to the reversibility of alignments.

Definition 2.16 (Total alignment, injective alignment) Given two ontologies o and o' , an alignment A over o and o' is called a *total alignment* from o to o' if and only if

$$\forall e \in Q_L(o), \exists e' \in Q_{L'}(o'); \quad \langle e, e', = \rangle \in A$$

and, it is called an *injective alignment* from o to o' if and only if

$$\forall e' \in Q_{L'}(o'), \exists e_1, e_2 \in Q_L(o); \quad \langle e_1, e', = \rangle \in A \wedge \langle e_2, e', = \rangle \in A \Rightarrow e_1 = e_2$$

These properties tightly depend on the ontology entity languages which are chosen for alignments.

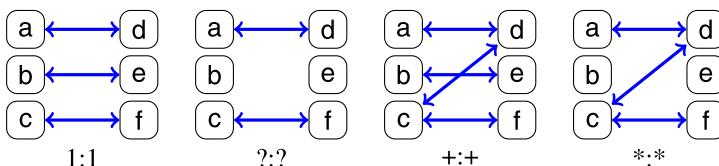
Usual mathematical properties apply to these alignments. In particular, a total alignment from o to o' is a surjective alignment from o' to o . A total alignment from both o and o' , which is injective from one of them, is a bijection. In mathematical English, an injective function is said to be *one-to-one* and a surjective function to be *onto*. Due to the wide use among matching practitioners of the term *one-to-one* for a bijective, i.e., both injective and surjective, alignment, we will only use one-to-one for bijective.

Finally, we can extend these definitions when correspondence relations are *not* equivalences. In such a case, they do not ensure the same properties. For instance, injectivity does not guarantee reversibility of the alignment used as a transformation.

In conceptual models and databases, the term multiplicity denotes the constraints on a relation. Usual notations are 1:1 (one-to-one), 1:m (one-to-many), n:1 (many-to-one) or n:m (many-to-many). If we consider only total and injective properties, denoted by ? for injective and total, + for injective, + for total and * for none, and the two possible orientations of the alignments, from o to o' and from o' to o , the multiplicities become: ?:?, ?:1, 1?:, 1:1, ?:+, +?:, 1:+, +:1, +:+, ?:*, *?:, 1:*, *:1, +:*, *:+, *:+ (Euzenat 2003).

Example 2.17 (Alignment multiplicity) The alignment of Example 2.14 is ?:?. If we add the correspondence Product \geq Volume, then it is ?:*. If we now consider relating any entity of the second ontology to another entity of the first one, then it becomes ?:+.

The four pictures below display some of the possible configurations for two ontologies composed of three classes each.



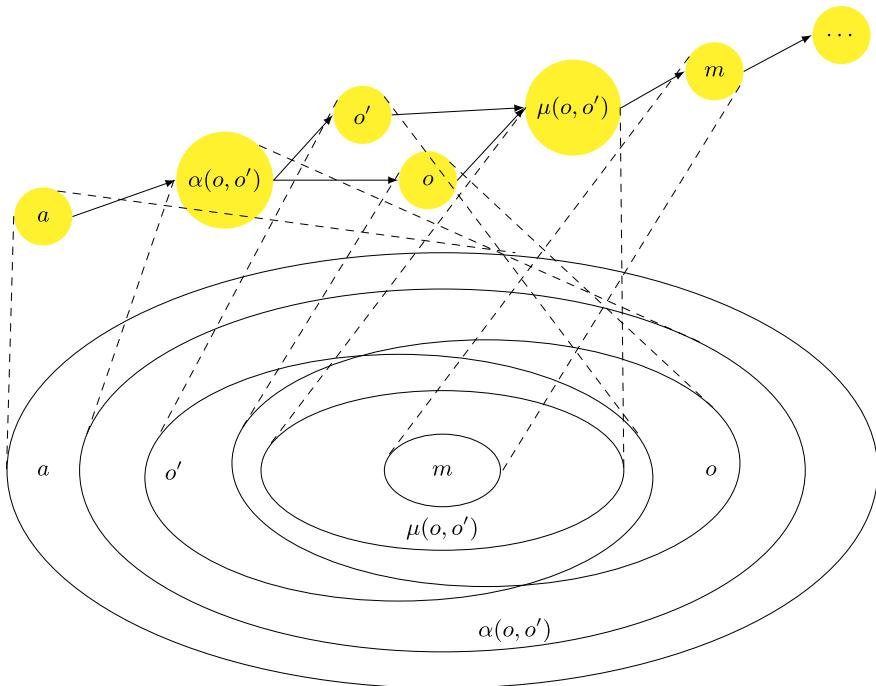


Fig. 2.10 Relations between ontologies, alignment $\alpha(o, o')$ and the corresponding model-theoretic interpretation. Each ontology in the top of the figure is represented by its set of models in the bottom. These ontologies correspond to an initial (possibly empty) ontology m (itself approximating some even more complete ontology). $\alpha(o, o')$ is an alignment of o and o' with regard to the way it approximates them and $\mu(o, o')$ is their merge with regard to the way o and o' approximate it.

2.5.3 Towards a Semantics for Matching and Alignments

As an introduction to the ideas behind matching, we present here a very simple yet powerful description. It relies on the principle that ontologies can approximate other ontologies and that ontologies to be matched are approximation of a common ideal ontology. We give the classical interpretation of it in both model-theoretic terms and categorical terms. This is informally presented in Fig. 2.10.

Let approximation be a relation between ontologies which expresses that one ontology a is a representation of at least the same modelled domains as another $\alpha(o, o')$. In logic, this relation corresponds to entailment. In category theory, the ontology is called an *object* and the approximation is called a *morphism*. One can define other relations between ontologies, such as having at least one common approximated ontology. Syntactically, it is possible to provide a set of generators that will complete an ontology, e.g., adding a constraint on a class, classifying an individual, providing an approximated ontology.

Model-theoretic semantics assigns to any ontology the set of its models (see Sect. 2.2.2). If the ontology is correctly designed, the modelled domain is part of these. Model-theoretic semantics provides a formal meaning to the intuitions behind such a notion as approximation: an ontology is approximated by another if its models are also models of the other: this is the standard interpretation of entailment. Thus, the more approximated an ontology, the fewer models it has.

In these very general terms, matching two ontologies o and o' consists of finding a most specific ontology $\alpha(o, o')$ that approximates both ontologies. If one ontology is approximated by another, the result of matching should be the latter $\alpha(o, o') = o$. In model-theoretic terms, it corresponds to finding an ontology whose set of models is maximal for inclusion and is included in the intersection of the set of models of two aligned ontologies. In categorical terms, this means that there exists an object $\alpha(o, o')$ and a pair of morphisms from it to the ontologies o and o' , and for every other object satisfying these conditions there exists a morphism from it to $\alpha(o, o')$.

This general description of alignments can be compared to three cases explaining reasons for the existence of conceptual heterogeneity that were introduced in Sect. 2.3. In fact, if we take these cases literally:

Coverage mismatch, corresponding to two ontologies modelling totally different domains, resorts to $\alpha(o, o') = \emptyset$, \emptyset being an empty ontology;

Granularity mismatch, corresponding to two ontologies modelling the same domain with different precisions, corresponds to the case where $\alpha(o, o') = o$, i.e., one ontology is an approximation of the other;

Perspective mismatch, corresponding to the representation of different aspects of the same domain at the same granularity, is the general case presented here.

Of course, in real-world matching tasks, the most frequent cases do not distinctly belong to one of these cases but rather mix them.

This general intuition about matching, similar to that of (Kalfoglou and Schorlemmer 2003b), strongly resembles a category-theoretic framework. Indeed, work towards more precise categorical definitions of ontology matching has been developed (Hitzler et al. 2005; Zimmermann et al. 2006; Kalfoglou et al. 2007). In particular, in categorical terms the merge corresponds to a push-out construction (Bench-Capon and Malcolm 1999; Alagic and Bernstein 2001) or its generalisation as a colimit (Hitzler et al. 2005). Such a categorical definition was used for designing matchers in (Atencia and Schorlemmer 2012). We provide a simple semantic foundation for alignments. It is useful to understand what is expected from a matching algorithm. However, the semantics of alignments defines how alignments must be interpreted and not how alignments must be found by a matching algorithm. In this respect, it is only a semantics for interpreting alignments and not for the matching operation.

The usual way of providing a semantics for related conceptual systems is through modal logic of knowledge and belief (Fagin et al. 1995; Wooldridge 2000). In line with the work on data integration, we only give a first-order model-theoretic semantics because, so far, networks of ontologies do not offer a language for expressing statements about knowledge and belief. It depends on the semantics of ontologies

(Sect. 2.2.2) but does not interfere with it. From a network of ontologies, the semantics of the whole system is defined with respect to the semantics of each individual ontology.

Given a network of ontologies, the first problem arising is the possible noncompatibility of their domains of interpretation. It is possible to consider different positions with regard to the domain of interpretation:

- (a) For all these ontologies, there is a unique domain of interpretation D . This approach is useful when ontologies describe a set of well defined entities, like the set of files shared in a peer-to-peer system. This approach has been taken in (Calvanese et al. 2002b, 2004).
- (b) Each ontology has a domain of interpretation disjoint from the others. This is especially useful for computational purposes as it allows ordered reasoning across connected ontologies. Such a formalism has been developed as \mathcal{E} -connections in which only specific relations can connect entities of the different ontologies (Kutz et al. 2004; Cuenca Grau et al. 2006).
- (c) For each ontology o , the domain D_o may be different. Domains are related with the help of domain relations $r_{o,o'}$ which map elements of D_o to corresponding elements of $D_{o'}$. This approach is used in (Borgida and Serafini 2003; Bao et al. 2009).
- (d) There is no constraint on the domain of interpretation of ontologies. For dealing with this assumption, a universal domain U is used, that may be defined as the union of all the domains under consideration, and an equalising function γ or rather a set of equalising functions: $\gamma_o : D_o \longrightarrow U$. This is the assumption that is considered in (Zimmermann and Euzenat 2006).

We call reduced semantics the technique which consists of merging the ontologies with the correspondences transformed into ontological statements (Meilicke et al. 2009). This technique interprets alignments under option (a). It only works when the ontologies are in the same language or through translating both ontologies and the alignment into a more classical logic.

(Zimmermann and Euzenat 2006) considers the implications of options (a), (c) and (d). The first edition of this book provided a semantic framework tied to the latter option. We now provide a more neutral and more general perspective.

For that purpose, correspondences are interpreted with respect to three features: a pair of models from each ontology and a semantic structure, denoted by Δ , which depending on the interpretations may be empty, be an equalising function or be a domain relation structure (Zimmermann 2013). Hence, the semantics of correspondences are given with respect to ontology models and the semantic structure supporting correspondences. The models have to be adequately extended to entity languages used in the alignments.

Definition 2.18 (Satisfied correspondence) A correspondence $\mu = \langle e, e', r \rangle$ is satisfied by two models m, m' of o, o' for some semantic structure Δ if and only if

$$\langle m(e), m'(e') \rangle \in r^\Delta,$$

such that r^Δ provides the interpretation of the relation r in the structure. This is denoted by $m, m' \models_\Delta \mu$.

Example 2.19 (Interpretation of relations) In the language used as example, if m and m' are respective models of o and o' and Δ interprets relations as their set-theoretic counterpart:

$$\begin{aligned} m, m' \models_\Delta \langle c, c', = \rangle &\quad \text{iff } m(c) = m'(c') \\ m, m' \models_\Delta \langle c, c', \leq \rangle &\quad \text{iff } m(c) \subseteq m'(c') \\ m, m' \models_\Delta \langle c, c', \geq \rangle &\quad \text{iff } m(c) \supseteq m'(c') \\ m, m' \models_\Delta \langle c, c', \perp \rangle &\quad \text{iff } m(c) \cap m'(c') = \emptyset \end{aligned}$$

This is typically what happens for option (a).

Definition 2.20 (Models of alignments) Given two ontologies o and o' and an alignment A between these ontologies, a model of this alignment is a triple $\langle m, m', \Delta \rangle$ with $m \in \mathcal{M}(o)$, $m' \in \mathcal{M}(o')$, and Δ a semantic structure, such that $\forall \mu \in A, m, m' \models_\Delta \mu$ (denoted by $m, m' \models_\Delta A$).

As usual, an alignment is said satisfiable or consistent if there are models of the aligned ontologies that can be combined in such a way that all of its correspondences are satisfied. Otherwise, the alignment is said inconsistent. It is said valid if any of the ontology model correspondences have this property. The satisfiable set of alignments is far larger than the set of valid ones. By analogy with ontologies, the coherence of an alignment is defined as the ability for any class to have instances, i.e., that no class interpretation is empty in all models.

Models of networks of ontologies extends models of alignments. They select vectors of compatible models of each ontology in the network in the style of (Ghidini and Giunchiglia 2001). Compatibility consists of satisfying all the alignments of the network.

Definition 2.21 (Models of networks of ontologies) Given a network of ontologies $\langle \Omega, \Lambda \rangle$ with $n = |\Omega|$, a model of $\langle \Omega, \Lambda \rangle$ is a $n + 1$ -uple $\langle m_1 \dots m_n, \Delta \rangle$ with $m_1 \in \mathcal{M}(o_1) \dots$ and $m_n \in \mathcal{M}(o_n)$, such that for each alignment $A \in \Lambda(o_i, o_j)$, $m_i, m_j \models_\Delta A$. The set of models of $\langle \Omega, \Lambda \rangle$ is denoted by $\mathcal{M}(\langle \Omega, \Lambda \rangle)$.

In that respect, alignments act as model filters for the ontologies. They select the ontology interpretations which are coherent with the alignments. This allows for transferring information from one ontology to another since reducing the set of models will entail more consequences in each aligned ontology.

From these definitions of models, it is possible to define the usual notions, such as consistency or consequences for an alignment or network of ontologies: a network of ontologies is *consistent* if it has at least one model, otherwise it is inconsistent.

We have introduced the notion of α -consequences as the correspondences which are consequences of a particular set of ontologies and alignments (Euzenat 2007).

Definition 2.22 (α -Consequence of networks of ontologies) Given a finite set of n ontologies Ω and a finite set of alignments Λ between pairs of ontologies in Ω , a correspondence μ between two ontologies o_i and o_j in Ω is an α -consequence of $\langle \Omega, \Lambda \rangle$ (denoted by $\models_{\Omega, \Lambda} \mu$ or $\langle \Omega, \Lambda \rangle \models \mu$) if and only if for all models $\langle m_1, \dots, m_n, \Delta \rangle$ of $\langle \Omega, \Lambda \rangle$, $m_i, m_j \models_{\Delta} \mu$.

The set of α -consequences of a network of ontologies is called its α -closure.

From the alignment semantics, it is possible to decide if an alignment is a consequence of another or if the alignment makes the set of ontologies and alignments inconsistent. This has been used for:

- deducing alignments, i.e., computing α -consequences for finding missing alignments between two ontologies or completing an existing alignments (see Sect. 6.5);
- checking alignment consistency and repairing alignments (see Sect. 7.8.2);
- evaluating alignments (see Sect. 9.3.2); and
- evolving alignments (see Sect. 12.8).

This definition coincides with a coherent model of the world in which all models satisfy all alignments. This is the standpoint of an omniscient observer and it corresponds to the global knowledge of a distributed system as defined in (Fagin et al. 1995).

However, if one ontology is inconsistent then the network of ontologies has no model. Therefore, even agents not connected to the inconsistent ontology cannot compute reasonable models. Moreover, an agent knowing an ontology and the related alignments would like to use the system by gathering information from its neighbours and considering only the models of this information. Thereby, it would be able to compute consequences through some complete deduction mechanisms. This is important when asking agents to answer queries and corresponds to local knowledge in (Fagin et al. 1995). This is the knowledge an agent can achieve by communicating only with the agents it is connected to in a network of ontologies.

From that standpoint, there can be several ways to select the acceptable models given the distributed system (here, $X \models_{\Delta} \lambda(o, o')$ is to be interpreted as $\forall A \in \Lambda(o, o'), X \models_{\Delta} A$):

$$\mathcal{M}_{\Omega, \Lambda}^0(o) = \mathcal{M}(o)$$

$$\mathcal{M}_{\Omega, \Lambda}^{\exists}(o) = \{m \in \mathcal{M}(o); \exists \Delta; \forall o' \in \Omega, \exists m' \in \mathcal{M}(o'); m, m' \models_{\Delta} \lambda(o, o')\}$$

$$\mathcal{M}_{\Omega, \Lambda}^{\exists*}(o) = \{m \in \mathcal{M}(o); \exists \Delta; \forall o' \in \Omega, \exists m' \in \mathcal{M}_{\Omega, \Lambda}^{\exists*}(o'); m, m' \models_{\Delta} \lambda(o, o')\}$$

$$\mathcal{M}_{\Omega, \Lambda}^{\uparrow\exists}(o) = \{m \in \mathcal{M}(o); \forall o' \in \Omega, \exists (\vec{m}, \Delta) \in \mathcal{M}(\langle \Omega, \Lambda \rangle); \vec{m}_o = m\}$$

$$\mathcal{M}_{\Omega, \Lambda}^{\forall*}(o) = \{m \in \mathcal{M}(o); \exists \Delta; \forall o' \in \Omega, \forall m' \in \mathcal{M}_{\Omega, \Lambda}^{\forall*}(o'); m, m' \models_{\Delta} \lambda(o, o')\}$$

$$\mathcal{M}_{\Omega, \Lambda}^{\forall}(o) = \{m \in \mathcal{M}(o); \exists \Delta; \forall o' \in \Omega, \forall m' \in \mathcal{M}(o'); m, m' \models_{\Delta} \lambda(o, o')\}$$

$$\mathcal{M}_{\Omega, \Lambda}^{\forall\forall}(o) = \{m \in \mathcal{M}(o); \forall o' \in \Omega, \forall \langle \vec{m}, \Delta \rangle \in \mathcal{M}(\langle \Omega, \Lambda \rangle); \vec{m}_o = m\}$$

These approaches have been ordered from the more optimistic to the more cautious. $\mathcal{M}_{\Omega, \Lambda}^{\exists}$ selects the models that satisfy each alignment in at least one model of the connected ontology. $\mathcal{M}_{\Omega, \Lambda}^{\forall}$ is very strong since all alignments must be satisfied by all models of the connected ontologies. $\mathcal{M}_{\Omega, \Lambda}^{\exists*}$ and $\mathcal{M}_{\Omega, \Lambda}^{\forall*}$ are fixed point characterisations that, instead of considering the initial models of the connected agents, consider their selected models by the same function. This contributes to propagating the constraints to the entire connected components of the network of ontologies. While for $\mathcal{M}_{\Omega, \Lambda}^{\exists*}$ this strengthens the constraints, for $\mathcal{M}_{\Omega, \Lambda}^{\forall*}$, this relaxes them with respect to $\mathcal{M}_{\Omega, \Lambda}^{\forall}$. Here, an inconsistent model is a problem only to related agents and only for versions $\mathcal{M}_{\Omega, \Lambda}^{\exists}$ and $\mathcal{M}_{\Omega, \Lambda}^{\exists*}$, which require the existence of a model for each related ontology. $\mathcal{M}_{\Omega, \Lambda}^{\forall\exists}$ and $\mathcal{M}_{\Omega, \Lambda}^{\forall\forall}$ are global versions, which consider models of the whole network of ontologies supporting the filtered models. The satisfaction of $\lambda(o, o')$ is entailed by the choice of m and $\vec{m}_{o'}$ in a model of the network.

Each of these options allows for specialising the semantics of ontologies in a network based on the models of networks of ontologies considered above. It is also analogous to the distributed knowledge of the system following (Fagin et al. 1995).

One can be even more restrictive by considering only a subset of the possible models of each ontology.

When dealing with ontology matching between a pair of ontologies, the matter of semantics between ontologies is not related to the alignment but to the interpretation of the full network of ontologies, for instance, depending on whether one wants to enforce global consistency or not. In this book we will not take a position on such a matter and will only retain the basic interpretation framework provided above.

Finally, such a formalism contributes to the definition of the meaning of alignments: it describes what are the consequences of ontologies with alignments, i.e., what can be deduced by an agent. However, it does not describe what the correct alignments are: matching is not a deductive task, but an inductive one. The framework is nevertheless particularly useful for deciding if delivered alignments are consistent, i.e., if networks of ontologies have a model or not. Hence, it is useful for specifying what is expected from matching algorithms and how they should be designed or evaluated.

2.6 Summary

In this chapter, we have first described different kinds of data and conceptual models and observed their expressivity. Formal ontologies turn out to be their most elaborate form. This means that, by developing and reviewing ontology matching, this book will cover other areas as well.

In fact, on the one hand, schema matching is usually performed with the help of techniques trying to guess the meaning encoded in the schemas. On the other hand, ontology matching systems primarily try to exploit knowledge explicitly encoded in the ontologies. In real-world applications, schemas and ontologies usually have both well defined and obscure terms and contexts in which they occur, therefore, solutions for both problems would be mutually beneficial. Consequently, we focus our attention on ontology matching as a task that comprises many characteristics of other forms of matching between data and conceptual models.

Then, we focussed on identifying what semantic heterogeneity is and why it requires matching. We have presented various reasons why mismatches may occur between ontologies. Their variety and the fact that they often occur together constrain to develop multiple approaches for matching ontologies. These techniques will be classified in Chap. 4 and further detailed in the forthcoming ones.

Finally, we have defined the action of matching ontologies and its result: the alignment. The interpretation of alignments has been provided first generally but informally, before introducing a semantics for it in networks of ontologies. This semantics should help the reader to understand how the results of ontology matching have to be interpreted and what is expected from ontology matchers.

Chapter 3

Methodology

Ontology matching is an important operation in modern ontology engineering because of the heterogeneous environments in which ontologies are designed, developed and supposed to work. Methodologically, it is worthwhile to express relations between ontologies since this allows for: (i) working with small and self-sufficient modules, instead of monolithic ontologies, (ii) expressing the links between two versions of the same ontology, and thus, updating data from one ontology to another, or (iii) putting back an ontology in the context of an upper-level ontology, allowing it to become more consensual with other ontologies of that domain.

Hence, a methodology for ontology matching is required and needs to be supported for helping engineers to develop applications. Yet, at present, almost no support exists for such an activity at the methodological or at the tool level. Even in the database field, where similar problems have been considered for years, there is no consensus methodology on how database schema matching may be conducted. The first edition of this book covered many facets of ontology matching, but not explicitly a methodology for it. Very few works have focussed on matching methodology: (Mochol 2009) dedicated a whole doctoral thesis to the topic of matcher selection; (Corcho 2005) considered more specifically the methodology for designing an ontology translation method, including a matcher; (Euzenat et al. 2008a) is more methodological, but not specifically focussed on the individual act of matching.

This chapter provides guidance for matching ontologies based on existing partial guidelines and overall experience collected so far in the field. We will illustrate this methodology through the fictitious example of a research library wanting to improve its services to consumers by offering its catalog on the web and linking it with other on-line resources, such as its national library or publisher data sources. For that purpose, it will have to identify these external resources and find or establish alignments between its own ontology (here the Bibliography ontology) and those of the external resources.

In what follows, we provide methodological guidelines for ontology matching following (Euzenat and Le Duc 2012). First, we consider the alignment life cycle, from which we derive a methodological workflow (Sect. 3.1). Then, this methodology is detailed in eight steps:

- define the characteristics of the concrete problem to solve (Sect. 3.2);
- find if available and suitable alignments exist for the given problem (Sect. 3.3);
- select or build a matcher if necessary (Sect. 3.4);
- run the matcher (Sect. 3.5);
- evaluate the obtained alignment (Sect. 3.6);
- improve it by reiterating the matching process (Sect. 3.7);
- document and share the satisfying alignment (Sect. 3.8); and finally
- process the alignment via a generator suitable for the given application task (Sect. 3.9).

3.1 The Alignment Life Cycle

Contrary to ontology engineering, which is ultimately an open (design) activity, ontology matching is an inductive task bounded by the ontologies to be matched. Hence, a methodology for it can be more focussed.

We do not consider ontology matching as an independent activity. Instead, we consider it as related to ontology management: when ontologies evolve, alignments must follow this evolution as well. Moreover, ontology matching should be considered in a dynamic perspective in which the result of matching has its own life cycle, which has to be maintained and evolved as well (Euzenat et al. 2008a). This is illustrated in Fig. 3.1, which shows the alignment life cycle. It takes into account the evolution of alignments as well as the importance of considering alignments as first class objects, which can be shared. As such, alignments can be manipulated to better suit the needs of users. We consider this ontology alignment life cycle and further investigate the methodology for supporting it. This methodology emphasises alignments as these are particularly useful in open environments (Suárez-Figueroa et al. 2012).

Based on the time at which ontology matching is supposed to take place, two different matching activities may occur. If ontology matching is supposed to occur at design time, then its goal is to match two ontologies for either using it immediately on static data or using it at run time under the form of a generated program, e.g., query mediator, message translator. In turn, if matching has to occur at run time, then the goal of the activity is to generate a matching process that achieves the results on-the-fly, for example, for composing semantic web services. In such a case, ontologies to be matched are not known in advance. The distinction between run-time and design-time ontology matching is important in practice because the output of these two operations is different.

When ontology matching is performed at design time, only the resulting alignment is available at run time: no more matching is necessary. So, there is no run-time constraint on matching. When it is performed at run time, no design-time alignment is available. So, the goal of the designer is to design a matching process instead of an alignment. In this case, run-time constraints (speed, memory) become important.

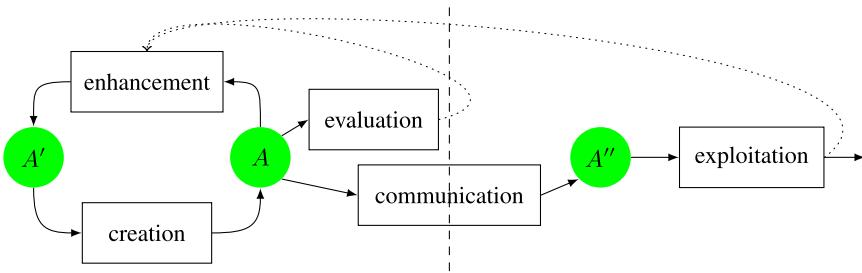


Fig. 3.1 The ontology alignment life cycle (adapted from (Euzenat et al. 2008a)). Alignments are first *created* through a matching process. Then they can go through an iterative loop of *evaluation* and *enhancement*. Evaluation consists of assessing properties of the obtained alignment. Enhancement may be obtained through manual change of the alignment, or through application of refinement procedures, e.g., selecting some correspondences by applying thresholds. When an alignment is deemed worth publishing, then it can be stored and *communicated* to other parties interested in such an alignment. Finally, the alignment can be *exploited*, e.g., transforming it into another form or interpreted for performing actions, like mediation or merging. Most of the actions of the life cycle may be performed either manually or automatically.

However, from the functional point of view, these two operations can also be seen as the same since, in practice, they generate an ontology matching process, which is executed at different moments. Hence, the methodology that we apply is the same in both cases because it consists of choosing software components and procedures, which are applied at different times.

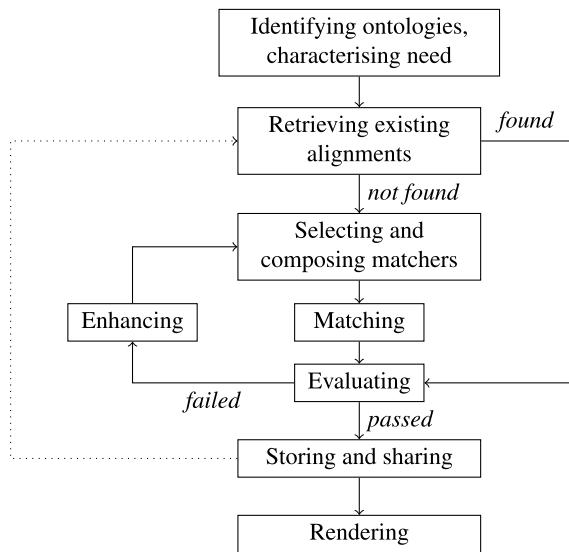
Our methodological guidelines are summarised by the workflow of Fig. 3.2. Each step of this workflow is described in the following subsections. The workflow addresses matching from a strictly technical perspective and it does not take into account all the needs, e.g., solution deployment, of an end-to-end service delivery. Such issues as the integration of this workflow in work process or data privacy issues are not addressed. It is assumed that these matters are dealt with independently.

3.2 Identifying Ontologies and Characterising Needs

The first task identifies the ontologies to be matched and characterises the need. Indeed, the type of required alignment is different if the goal is to merge two ontologies in a knowledge-based system or to add yet another data source to a query mediator. In the former case, the alignment will have to be strictly correct—otherwise the system may draw incorrect inferences—but the relationships can be diverse: subsumption and disjointness assertions may be useful. In the latter case, lack of completeness is not a problem—because other sources may return the missing answers—but relations other than equivalence are not straightforwardly used in query mediation.

It is also useful to characterise the kind of ontologies: Are they labelled in the same natural language? What is their expressiveness? Are they populated by available individuals?

Fig. 3.2 The matching methodology workflow. It goes step-by-step through *characterising* the problem, *selecting* existing alignments, *selecting* appropriate matchers, *running* the matchers, *evaluating* the results and correcting the choices made before (matchers, parameters), *documenting* and publishing good results, and finally *using* them.



Characterising the situation in which matching will be performed should not be neglected. It will determine the choice of matchers or alignments as well as the parameters to care for. From the discussion of Chap. 1, different requirements were identified for different applications. These requirements concern:

- the type of available input a matching system can rely on, such as schema or instance information. There are cases when data instances are not available, for instance, due to security reasons (Clifton et al. 1997) or when there are no instances given beforehand. Therefore, these applications mandate a matching solution that can work without instances (here schema-based method).
- some specific behaviour of matching, such as requirements of (i) being *automatic*, i.e., not relying on user feedback; (ii) being *correct*, i.e., not delivering incorrect matches; (iii) being *complete*, i.e., delivering all the matches; and (iv) being performed at *run time*.
- the use of the resulting alignments as described above. In particular, how the identified alignment is going to be processed, e.g., by merging the data or conceptual models under consideration or by translating data instances among them.

Table 3.1 summarises what we found to be the most important requirements for the matching operation in the applications considered in Chap. 1. This is obviously a general approximation that must be adapted to each particular application.

Some of these hard requirements can be derived into comparative (or nonfunctional) requirements, such as speed, resource consumption (in particular memory requirements), degree of correctness or completeness. They are useful for comparing solutions on a scale instead of an absolute (yes/no) comparison. Moreover, they allow trading a requirement, e.g., completeness, for another more important one,

Table 3.1 Summary of application requirements.

Application	instances	run time	automatic	correct	complete	operation
Ontology evolution (Sect. 1.1)	✓			✓	✓	transformation
Schema integration (Sect. 1.2)	✓			✓	✓	merging
Catalogue integration (Sect. 1.2)	✓			✓	✓	data translation
Data integration (Sect. 1.2)	✓			✓	✓	query mediation
Linked data (Sect. 1.3)	✓			✓		data interlinking
P2P information sharing (Sect. 1.4)		✓				query mediation
Web service composition (Sect. 1.5)		✓	✓	✓		data mediation
Multiagent communication (Sect. 1.6)		✓	✓	✓	✓	data translation
Context matching in ambient computing (Sect. 1.6)		✓	✓	✓		data translation
Semantic web browsing (Sect. 1.7)	✓	✓				navigation
Query answering (Sect. 1.7)	✓	✓		✓		query reformulation

e.g., speed. These general requirements for applications will be used in Chap. 8 for assessing the capacity of matching systems to be applied to particular applications, in Chap. 9 for designing evaluation procedures related to applications, and in Chap. 12 to classify the operations performed after matching.

Example 3.1 (Characterising matching needs) The requirements for the library application are, on the one hand, to have alignments with other standard bibliographic ontologies so that queries may be translated and data can be exported to these ontologies, and, on the other hand, to have an alignment with the web indexing ontology schema.org. Matching will be performed off-line, without time constraints. The ontologies, having been developed independently and for different purposes, are not expected to match exactly. Correct correspondences are expected, completeness is secondary. More precisely, on the first type of ontology, correctness is very important, but for schema.org completeness is more important. The types of operation to be performed with the resulting alignments are data export (for exposing data as linked data and annotating web pages in RDFa) and query transformation (for interrogating remote resources).

The source ontology is bibo (the Bibliography ontology). It offers a comprehensive description of publications and documents of various origins, featuring classes such as Magazine, Interview, LegalDocument together with Thesis. In turn, Schema.org is an ontology designed for annotating web pages, so that web search engines are aware of their content. It is thus a very general ontology not specific to the domain.

In such a setting, Watson (d'Aquin and Motta 2011) and LOV¹ (Linked Open Vocabularies) allow for finding further ontologies that may be useful. These are (see Table 3.2):

¹<http://lov.okfn.org>.

- bibtex: an ontology defined from the BibTeX tool that expresses bibliographies used in scientific articles,
- frbr (Functional Requirements for Bibliographic Records): an ontology designed by the International Federation of Library Associations, which is widely used in linked data produced by national libraries. It redefines the Dublin core annotations,
- biblio: another citation-oriented bibliographic ontology that was derived from bibo, and so can be expected to share many common concepts with it.

Other ontologies are available, such as upper-level ontologies or the bibliographic ontologies used in the Ontology Alignment Evaluation Initiative (OAEI) campaigns (Sect. 9.1.3).

A quick study of these ontologies reveals the following characteristics:

Table 3.2 Characteristics of the considered ontologies.

Ontology	URL	Lang.	Form.	Classes	Relations	Properties
bibo	http://purl.org/ontology/bibo/	OWL	en	69	51	55
schema.org	http://schema.org/	OWL	en	295	113	69
bibtex		OWL	en	13	5	27
biblio	http://purl.org/net/biblio	OWL	en	57	1	0
frbr	http://purl.org/vocab/frbr/core	OWL	en	73	52	6

The ontologies to be matched are all in OWL and written in English. Beside the bibtex ontology, which is smaller, and schema.org, which is larger, these ontologies have a comparable size and structure (they define very few properties). It will be necessary to use matchers able to handle such characteristics, and multilingual matchers are not mandated.

The type of mismatch between these ontologies is expected to be different: schema.org is a broad general-purpose ontology, so matches are rather expected at the higher level of ontologies. On the contrary, bibtex is rather small and focussed, so only a part of the information may overlap.

3.3 Retrieving Existing Alignments

Retrieving existing alignments which satisfy the need of the application is the second step. Alignments can be found on the web or through specialised directories. Reusing existing alignments should be privileged because of the cost of generating such alignments and their presumably good quality. For that purpose, the sharing step (see Sect. 3.8) prepares alignment retrieving.

Ideally, alignments should come with annotations characterising their level of trustworthiness, the purpose for which they were built and the type of relations they use.

These alignments must concern the ontologies to be matched and have to satisfy the alignment constraints established in Sect. 3.2. In particular, correctness and completeness are criteria to use for selecting among various alignments. These criteria may be assessed manually, on a sample, or can be inferred through the properties of their generation methods. In particular, metadata attached to such alignments may reveal the method used for matching the ontologies (in particular, if these are automatic or manually generated alignments); they can cover manual assessments about the alignment (people publishing them can annotate the alignments to tell what they are good for); or they may contain indications of their intended use which can be matched with that of the current situation. Alignment metadata is discussed in detail in Sects. 3.8 and 10.2.

In practice, selecting an alignment requires:

- finding alignment repositories;
- retrieving the alignments between the ontologies to be matched;
- assessing the capacity of these alignments to address the needs previously identified, based either on metadata, or on the content of the alignments;
- choosing one alignment based on the assessment.

If apparently suitable alignments are available, users may directly move to the validation step (Sect. 3.6). Otherwise, it is necessary to create a new alignment from the ontologies, as explained in Sect. 3.4.

Example 3.2 (Retrieving existing alignments) Retrieving available alignments can be achieved by using an alignment server (Sect. 12.7). In the present case, many alignments to bibliographic ontologies can be found due to OAEI (Sect. 9.1.3), but none to the ontologies of interest. Querying LOV reveals that there exists one single link between these ontologies that may be interpreted as a correspondence:

`(biblio:Person, frbr:Person, owl:sameAs)`

It will thus be necessary to match these ontologies.

3.4 Selecting and Composing a Matcher

In order to build a new alignment, a suitable matcher has to be found. Many matching systems have been developed over the years. Users can choose among those presented in Chap. 8. They provide different results depending on the different types of data sets and matching contexts. Hence, the criteria elicited in the characterisation phase (Sect. 3.2) are also used for selecting the most appropriate matcher. There have been a few studies about how to choose a matcher depending on the characteristics of the ontologies and those of the expected alignments. (Euzenat et al. 2006)

provides a simple method for weighting matcher capabilities (speed, automaticity, precision, and recall as measured in matcher evaluations) against the application requirements defined as the answers to the questions of Sect. 3.2. The work of (Mochol 2009) uses a deep classification of matchers and the matching context in order to assess which matcher will be more adapted to a particular context. This assessment is made using the Analytic Hierarchy Process (AHP), which guides the decision process of choosing a matcher. It can work in automatic or manual mode. The OntoMas system (Huza et al. 2006) has been developed for helping and teaching how to carry out matching. For choosing a matcher, it processes a set of symbolic rules over a classification of tools and a characterisation of tasks.

The problem of such methods is that they require extensive information about available matchers, which is not always available, nor always accurate, when the assessment comes from matcher developers. An important source of information is the result of the various evaluation campaigns that have been run, most notably the OAEI campaigns (Sect. 9.1.3). They have evaluated many matchers in a variety of situations. Such results can be taken into account when choosing a matcher.

In practice, choosing a matcher can be achieved by:

- finding available matchers;
- assessing their capacity to generate alignments that fill the identified requirements, by reading their documentation or comparing their performances on similar tasks within evaluations;
- choosing one matcher based on this assessment.

It is possible to select several matchers and to compose them as a single matcher (Sect. 7.2). However, this is not the goal of this methodology to build a matcher. This book should serve this purpose. Other works try to automate this step, or the selection of matcher parameters (Sect. 7.6); they can be exploited in run-time matching processes.

Example 3.3 (Selecting a matcher) Users then proceed by selecting a matcher suited for matching these ontologies. In this case, given that ontologies are about a very close domain, and that they are written in the same natural language, users can select simple matchers based on string comparisons, such as a simple use of WordNet, to match terms from linguistic cues.

If this does not work sufficiently well, there are many more elaborate matchers available (see Chap. 8); the best way is to try them and to see the results (see Example 3.4). We will consider matchers that may be used easily, such as AROMA, which looks for association rules between concepts before extracting an alignment between them (Sect. 8.3.14) or LogMap, which interpret ontologies semantically (Sect. 8.3.26), and which had good results in the OAEI 2012 campaign (Aguirre et al. 2012).

Instead of merely composing a matcher, it is possible to learn how to match (Sect. 7.5). The methodology is then slightly derogatory from that of Fig. 3.2; it proceeds to:

- select a sample of the ontologies to match and decide what is the expected alignment between the entities of this sample;
- train a machine learning algorithm on this sample;
- run the resulting matcher on the full ontologies.

3.5 Matching Ontologies

The next step consists of running the selected matchers against the ontologies and collecting the resulting alignment. It may seem like the simplest step, methodologically speaking, because matchers have been designed exactly for this purpose.

However, users should not hesitate to run the matcher several times or to run several matchers, trying different sets of parameters and different thresholds. It is also useful to process matching incrementally, by curating the returned alignment and feeding it again to the matcher for improving the results. In fact, all the procedures that can be applied at the enhancing phase (Sect. 3.7) can also be directly applied during the matching phase without any prior evaluation. Hence, this step can be further decomposed into a more complex subworkflow (see Fig. 3.3). Section 3.7 provides some refinements of the matching workflow.

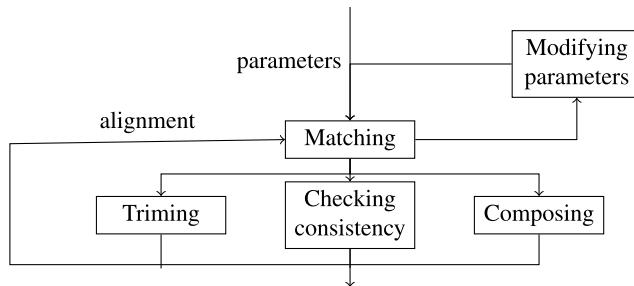


Fig. 3.3 The subworkflow of fine-tuning matchers (all tasks but Matching are optional). After *matching*, it is possible to apply automatically some alignment manipulation that *trims* the alignment under a threshold, *checks* and *restores* the consistency of an alignment or *composes* the alignment with another alignment. The result of these manipulations can be fed back as input to the Matching operation or can be the final result of the workflow. Alternatively, it is possible to *modify* the parameters of the matcher and to run it again. These operations can be triggered manually or automatically.

Example 3.4 (Matching ontologies) The simple StringDistAlignment method with different string distances is run and results are displayed in Table 3.3.

First a string equality comparison is processed for identifying entities bearing the same name. The correspondences between bibo:Map and schema:Map or between bibo:editor and bibtex:editor, for instance, are correct. As expected, there are

Table 3.3 Overview on the matched entities. In parentheses are indicated the numbers of additional matches compared with the bibo ontology depending on matching method and threshold. The horizontal line separates simple matchers and more elaborate ones. * means that the system reported parsing errors.

Method	Threshold	schema	biblio	bibtex	frbr
String equality	1.	9	30	8	6
Levenshtein	.9	10 (+1)	30 (+0)	8 (+0)	6 (+0)
–	.8	10 (+1)	30 (+0)	11 (+3)	7 (+1)
SMOA	.9	21 (+12)	35 (+5)	16 (+8)	7 (+1)
–	.95	11 (+2)	31 (+1)	11 (+3)	6 (+0)
WordNet	.9	17 (+8)	39 (+9)	15 (+7)	13 (+7)
–	.95	13 (+4)	37 (+7)	13 (+5)	11 (+5)
AROMA	–	27	0*	1*	7
LogMap	–	8	56	3	12

more correspondences with biblio than with any other ontology, including the larger schema.org.

For finding more matches, three slightly more complex methods were used: the Levenshtein measure (edit distance, Sect. 5.2.1), the SMOA measure, which tries to better interpret the way people label things, e.g., by using syntactic variations (Sect. 5.2.1), and a WordNet-based measure, which depends on the number of synsets that terms have in common (Sect. 5.2.2). All three methods were first run with a threshold of .9. As can be seen, Levenshtein tends to return very few additional matches, while SMOA and WordNet return too many of them, e.g., matching bibo:Image to frbr:Person. Hence thresholds have been adapted to .8 for the former and .95 for the latter.

3.6 Evaluating Alignments

Once an alignment has been obtained, it is necessary to perform screening and validation. Evaluation can be applied on alignments that have been retrieved as well (Sect. 3.3). This task corresponds to the *evaluation* task of Fig. 3.1. We consider here what is specific to alignment evaluation during the matching activity. The evaluate/enhance loop in Fig. 3.2 corresponds to the feedback after *evaluation* in Fig. 3.1.

Evaluation consists of assessing properties of the obtained alignment. It can be performed either manually or automatically. Manual evaluation can be achieved by running a dry test of the final application or by asking an independent expert to assess the quality of the alignment and perform some manual assessment. For that purpose, graphical tools which allow users to navigate quickly both in the alignment and in the ontologies are invaluable (see Sect. 11.4).

An often overlooked functionality of matching algorithms is their ability to explain the provided alignments (Sect. 11.3). Explanations may be obtained by interacting with the matcher or by accessing metadata about a stored alignment.

Automated quantitative evaluation can be performed by using techniques for evaluating alignments presented in Chap. 9. These require extracting samples from the results and computing measures like precision and recall, which would provide an approximation of correctness and completeness. (Tordai 2012) came up with a methodology similar to this and insisted on sampling and evaluating partial results, e.g., across matchers, in order to improve the process.

There is no definitive answer as to what is a good result for evaluation. The evaluation must be performed so as to assess evaluation criteria. The characterisation of the problem (Sect. 3.2) aims at defining such success criteria. For some applications high recall is required, while for some others recall is not important. Moreover, the meaning of ‘high’ is not the same for all applications: a critical application which can break if some correspondences are missing will require 100 % recall, while a non-critical application may be satisfied with 80 %.

If the evaluation results are positive, i.e., the alignment satisfies these success criteria, then the obtained alignment can go through the next step, namely of storing and sharing (Sect. 3.8); otherwise, the alignment can be improved (Sect. 3.7) before being input to same or another matcher with the same or different parameters.

Example 3.5 (Evaluating alignments) There is no automatic way to evaluate the results of Example 3.4. They have to be manually looked into by users to assess their quality (they can be displayed by alignment editors).

Concerning the bibliographic ontologies, it is easy to first evaluate the matching of entities with the same name and then to further scrutinise those which are new using a `diff` tool. Using common evaluation measures, such as precision and recall, with respect to the result given by string equality can help in detecting if the alignment preserves the known correct correspondences provided by string equality.

Considering the matching with `frbr`, the purely string-based methods (Levenshtein, SMOA) are not very good. They preserve the correct correspondences found by string equality, but return very few additional correspondences. Such additional correspondences are `(bibo:translation, frbr:translator, =)` (Levenshtein) and `(bibo:Workshop, frbr:Work, =)` (SMOA), which are both incorrect. The WordNet-based method is more accurate in this case: it finds relations between `bibo:section` and `frbr:part`, which is clearly correct, and `bibo:content` and `frbr:subject`, which requires discussion. However, further lowering the threshold for this matcher provides incorrect matches, such as between `bibo:Person` and `frbr:Image`.

These very few additional correspondences suggest trying more elaborate techniques. Unfortunately, their performances are not better than those of the WordNet-based matcher: AROMA finds a relation between `bibo:Article` and `frbr:ScholarlyWork`, which is correct, as well as an incorrect one between `bibo:Report` and `frbr:Event`. LogMap does not do better: its high number of correspondences is due to that, contrary to the other matchers, it matches entities from external ontologies (thus finding that `foaf:Person` is equivalent to itself).

These results are not as bad as they may seem. In fact, frbr is a very abstract ontology and bibo is a very concrete one. Moreover, they already share many common entities specified in the Dublin core ontology. Hence, they do not share much more than what has been identified. This is different for the very close bibo and biblio leading to a higher number of matches. In this case, on top of the 30 literal matches, the WordNet-based method is able to find relations between bibo:Thesis and frbr:Dissertation or between bibo:Image and frbr:Figure.

3.7 Enhancing Alignments

Enhancement can be obtained either through manual modifications of the alignment, e.g., with the help of an alignment editor (Sect. 11.4), or through the application of refinement procedures, e.g., selecting correspondences by applying thresholds. This enhancing task can lead to:

- the selection of another matcher, as in Fig. 3.2, by going back to Sect. 3.4;
- the selection of another set of parameters to use with the same matcher, as in Fig. 3.3 (see Sect. 7.6); or
- the manipulation of the alignment through trimming under a particular threshold or combining several alignments, as in Fig. 3.3.

Among these procedures, the most straightforward one consists of trimming the alignment under some thresholds. There are many different ways to apply automatic thresholds (Sect. 7.7.1). Double thresholding (Lambrix and Liu 2009) uses two thresholds: above the upper threshold, correspondences are selected; under the lower threshold, they are discarded and the remaining correspondences are brought to the attention of users.

Consistency may also be restored when the resulting alignment has been found inconsistent in the evaluation (Sect. 3.6). By consistency checking, we do not necessarily mean logical consistency checking, but rather any violation of specific constraints on alignments (see Sect. 7.8). Enhancing may then consist of selecting a subset of the correspondences in an alignment which satisfies the constraints.

Alignments obtained from various sources, such as other matchers or alignment libraries, may be composed into a single alignment through various operators: composition, meet, join, union.

Example 3.6 (Enhancing alignments) Enhancement can be achieved by two means: either by manual editing of the resulting alignment or by running a new matcher, using new parameters or applying different thresholds to the results. The latter is what has been done by using different thresholds and by testing the more elaborate matchers, i.e., starting from Example 3.3. Both means can be interleaved: it is possible to edit an alignment and to use it as further input for a matcher.

Once this is achieved, picking up the few additional correct correspondences found by various matchers would likely provide a reasonably complete alignment. In principle, the resulting alignment can be checked for consistency. However, the ontologies are not expressive enough for finding inconsistency. Disambiguation may be applied since it would force dropping unwanted correspondences, e.g., `bibo:Image` to `frbr:Person`, because `bibo:Image` is already matched to `frbr:Image`.

3.8 Storing and Sharing

The process does not end once an alignment is deemed usable. The next step is to save and share it in a declarative format and to give it proper annotations to record its provenance and purpose. This step is very often overlooked, but it is vital if one wants to retrieve alignments in the corresponding step (Sect. 3.3): carefully annotating alignments will help others to reuse them. This task corresponds to the *communication* task of Fig. 3.1 and the dotted arrow in Fig. 3.2 corresponds to the availability of stored alignments after communication.

Storing an alignment requires some type of persistent storage. This is usually achieved through the use of a database management system, but a web site based on a file system may be sufficient as well. However, alignments must be properly indexed so that they can be retrieved, if necessary, on various characteristics (one ontology, pairs of ontologies, arity, etc.). Indexing may be direct, through URIs identifying alignments, or indirect, through queries looking for alignments based on their metadata. In general, it is preferable that both access modes be available. Finally, these alignments may be shared by interested communities. For that purpose, they should be accessible on the web. There are several systems supporting sharing alignments on the web (see Sect. 12.7).

Example 3.7 (Sharing alignments) Once an alignment of sufficient quality is established, especially if it has been curated by hand, it must be better documented, for instance by adding metadata explaining how it has been obtained, who has curated it and what is the confidence reached in each correspondence (see Sect. 10.2). In our example, relevant metadata can be as follows:

```
dc:creator Your name here (ACME Library)
dc:date 2013-04-12
dc:rights http://creativecommons.org/licenses/by/3.0/
align:method Manually assembled (JWNL confidence)
omwg:purpose Data export and query mediation
align:properties Checked for correctness
```

Then, it can be uploaded to an Alignment server so that it be visible to other people (see the previous step of Sect. 3.3).

3.9 Rendering and Processing Alignments

Finally, the alignment is transformed into another form or interpreted for performing actions like mediation or merging (see Chap. 12).

This task corresponds to the *exploitation* activity of Fig. 3.1. It is the natural outcome of matching. The exploitation of the alignment may be denoted by a different activity name, e.g., ontology merging or query transformation, taking directly the alignment as input. However, it may happen that ontology matching is considered as an activity in itself. In this case it will deliver its output in an appropriate format for another task. This is what is called ‘rendering’.

Rendering may deliver the alignment, as such, in RDF. It could then be later processed by an interpreter such as a query mediator. But it also can be transformed, as discussed in Sect. 10.1, into OWL axioms, SKOS relations or sets of `owl:sameAs` statements.

The dotted arrow in Fig. 3.1 expresses the feedback after using the alignment which may contribute enhancing it.

Example 3.8 (Rendering and processing alignments) The bibliographic descriptions stored in the local library will be exploited in various ways, such as being offered to third parties as linked data and being accessible through SPARQL querying.

On the one hand, they can be exported as HTML pages that provide semantic annotations. The information in descriptions will be rendered in RDF using the schema.org ontology and the resulting RDF will be embedded in the web pages through RDFa. This may be achieved either by generating an XSLT transformation applying to the data expressed in XML or by generating a query mediator for the database which poses the query with respect to bibo and constructs a resulting RDF graph with respect to schema.org according to the alignment.

On the other hand, if the library customers, or the applications they use, query the library database in SPARQL with respect to the bibo ontology, it is possible to give them access to data sources expressed in the other ontologies with respect to which bibo is matched. To that extent, mediators are generated that take a query expressed with respect to bibo, and transform it into a query in the other ontologies that can be submitted to remote SPARQL endpoints.

3.10 Summary

We think that methodological guidelines are more useful and better accepted if they are supported by tools, rather than delivered as rules to be applied. So far, existing support is available in the alignment manipulation part rather than the requirement analysis part. Some supporting systems are presented in Chap. 10.

Establishing relations between ontology entities is part of modern ontology engineering and an important task for engineering networks of ontologies (Suárez-Figueroa et al. 2012). This task remains difficult though there are many solutions

for carrying it out. We proposed methodological guidelines for ontology matching, which are integrated with the alignment life cycle and can cooperate with ontology engineering methodologies. In particular, we paid special attention to alignment sharing and reuse. This methodology is based on research on particular subtasks: some of these have been investigated in depth and others in less detail. Similarly, some tools cover parts of this methodology but none is able to support it entirely.

Hence, more work is necessary to achieve a fully instrumented ontology matching methodological support, and no doubt this will raise some demands for improvement in the proposed methodology.

Part II

Ontology Matching Techniques

Chapter 4

Classifications of Ontology Matching Techniques

Having defined what the matching problem and the process for solving it are, and before scrutinising further the details of matching techniques, we classify them from different standpoints. This should help better understanding these systems.

The major contributions of the previous decades are presented in (Larson et al. 1989; Batini et al. 1986; Kashyap and Sheth 1996; Parent and Spaccapietra 1998). Later, the topic has been surveyed in (Rahm and Bernstein 2001; Wache et al. 2001; Kalfoglou and Schorlemmer 2003b). The topic was also treated in the context of data integration or ontology change (Choi et al. 2006; Flouris et al. 2008; Bellahsene et al. 2011; Doan et al. 2012). These addressed the matching problem from different perspectives (artificial intelligence, information systems, databases) and analysed disjoint sets of systems. (Shvaiko and Euzenat 2005) considered the above mentioned works together, focussing on schema-based matching methods, and aiming to provide a common conceptual basis for their analysis. Here, we follow and extend this work on classifying matching approaches.

In this chapter, we first consider various dimensions along which a classification may be designed (Sect. 4.1). We then present our classification based on several of these dimensions (Sect. 4.2) and the classes of matching techniques (Sect. 4.3). Finally, we discuss some alternative classifications of matching approaches that have been proposed so far (Sect. 4.4).

4.1 Matching Dimensions

There are many independent dimensions along which algorithms may be classified. Following the definition of the matching process in Fig. 2.8, we may primarily classify algorithms according to (i) the input of the algorithms, (ii) the characteristics of the matching process, and (iii) the output of the algorithms. The other characteristics, such as parameters, resources, and input alignments, are considered less important. Let us discuss these three main aspects in turn.

4.1.1 Input Dimensions

These dimensions concern the kind of input on which algorithms operate. As a first dimension, algorithms can be classified depending on the data or conceptual models in which ontologies are expressed. For example, the Artemis system (Sect. 8.1.6) supports the relational, object-oriented, and entity–relationship models; Cupid (Sect. 8.1.11) supports XML and relational models; QOM (Sect. 8.3.3) supports RDF and OWL models. A second possible dimension depends on the kind of data that the algorithms exploit: different approaches exploit different information in the input ontologies. Some of them rely only on schema-level information, e.g., Cupid (Sect. 8.1.11), COMA (Sect. 8.1.12); others rely only on instance data, e.g., GLUE (Sect. 8.2.5); and others exploit both schema- and instance-level information, e.g., QOM (Sect. 8.3.3). Even with the same data models, matching systems do not always use all available constructs, e.g., S-Match (Sect. 8.1.18), when dealing with attributes, discards information about data types and uses only the attributes names. Some algorithms focus on the labels assigned to the entities, some consider their internal structure and the types of their attributes, and others consider their relations with other entities (see next section for details).

More generally, we can consider the information *origin* as a dimension, on which matching is based: this information can come directly from the content of the ontologies to be matched or from relations between the ontologies and other external resources, called *context*. Hence, this origin dimension can be split into *internal* or *content-based matching* and *external* or *context-based matching*. External resources can be formal, such as other ontologies, or informal, such as a collection of pictures annotated by the ontology or a thesaurus describing the terms used in the ontology. Relations with such external resources can be explicit, e.g., there is already an alignment with an external ontology or links to pictures, or implicit, i.e., such relations have to be established.

4.1.2 Process Dimensions

A classification of the matching process could be based on its general properties, as soon as we restrict ourselves to formal algorithms. In particular, it depends on the *approximate* or *exact* nature of its computation. Exact algorithms compute the precise solution to a problem; approximate algorithms sacrifice exactness for performance (Ehrig and Sure 2004). All of the techniques discussed in the remainder of the book can be either approximate or exact. Another dimension for analysing matching algorithms is based on the way they interpret the input data. We identify two categories depending on whether the matcher considers the input intrinsically or through some semantic theory of the considered entities. We call these categories: *syntactic* vs. *semantic* and discuss them in detail in the next section.

4.1.3 Output Dimensions

Apart from the information that matching systems exploit and how they manipulate it, the other important class of dimensions concerns the form of the result that these systems produce. The form of the alignment might be of importance: is it a one-to-one alignment between the ontology entities? Has it to be a final correspondence? Is any relation suitable?

Some other significant distinctions in the output results have been indicated in (Giunchiglia and Shvaiko 2003). One dimension concerns whether systems deliver a graded answer, e.g., that the correspondence holds with 98 % confidence or 4/5 probability, or an all-or-nothing answer, e.g., that the correspondence definitely holds or not. In some approaches, correspondences between ontology entities are determined using distance measures. This is used for providing an alignment expressing equivalence between these entities. Another dimension concerns the kind of relations between entities a system can provide. Most of the systems focus on equivalence ($=$), while a few others are able to provide a more expressive result, e.g., equivalence, subsumption (\leq), and incompatibility (\perp) (Giunchiglia et al. 2004; Bouquet et al. 2003b; Hamdi et al. 2010b; Spiliopoulos et al. 2010).

In the next section, we present a classification of techniques that draws simultaneously on these criteria.

4.2 Classification of Matching Approaches

4.2.1 Methodology

To ground and ensure a comprehensive coverage for our classification we have analysed state-of-the-art approaches used for ontology matching. Chapter 8 reports a partial list of systems which have been scrutinised pointing to (some of) the most important contributions. We have used the following guidelines for building our classification:

Exhaustivity: The extension of categories dividing a particular category must cover its extension, i.e., their aggregation should give the complete extension of the category.

Disjointness: In order to have a proper tree, the categories dividing one category should be pairwise disjoint by construction.

Homogeneity: In addition, the criteria used for further dividing one category should be of the same nature, i.e., should come from the same dimension introduced in Sect. 4.1. This usually helps guarantee disjointness.

Saturation: Classes of concrete matching techniques should be as specific and discriminative as possible in order to provide a fine-grained distinction between possible alternatives. These classes have been identified following a *saturation* principle: they have been added and modified until saturation was reached, i.e., taking

into account new techniques did not require introducing new classes or modifying them.

Disjointness and exhaustivity of the categories ensure the stability of the classification, namely that new techniques will not occur in between two categories. Categories of matching approaches represent the state of the art. Obviously, with appearance of new techniques, they might be extended and further detailed.

The exact vs. approximate opposition has not been used because each of the methods described below can be implemented as exact or approximate algorithms, depending on the goals of the matching system.

We build on the previous work on classifying automated schema matching approaches of (Rahm and Bernstein 2001) which distinguishes between *elementary* (individual) matchers and *composition* of matchers. Elementary matchers comprise *instance- and schema-based, element- and structure-level, linguistic and constraint-based* matching techniques. *Cardinality* and *auxiliary information*, e.g., thesauri, global schemas, can also be taken into account.

For classifying matching techniques, we introduced two synthetic classifications in (Shvaiko and Euzenat 2005), based on what we have found to be the most salient properties of the matching dimensions. These two classifications are represented by two trees sharing their leaves. The leaves represent classes of matching techniques and their concrete examples. In this edition of this book, we revised this classification based on the evolution of the state of the art, in particular the development of new approaches, which have changed the balance with respect to the classification of the first edition, e.g., context-based matching. These two revised and updated synthetic classifications are (see Fig. 4.1):

- *Granularity/Input interpretation* classification based (i) on the matcher granularity, i.e., element- or structure-level, and then (ii) on how the techniques generally interpret the input information (Sect. 4.2.2),
- *Origin/Kind of input* classification based (i) on the origin of the information considered by the matcher, and (ii) on the kind of input taken into account by matching techniques (Sect. 4.2.3).

The overall classification of Fig. 4.1 can be read both in descending (focussing on how the techniques interpret the input information) and ascending (focussing on the origin of matching clues before their interpretation) manner in order to reach the layer of *Concrete techniques*. It is designed in a way that offers a planar graph layout.

4.2.2 Granularity/Input Interpretation Layer

Matchers are distinguished by the *Granularity/Input interpretation* layer according to the following classification criteria:

- *Element-level vs. structure-level*: Element-level matching techniques compute correspondences by analysing entities or instances of those entities in isolation,

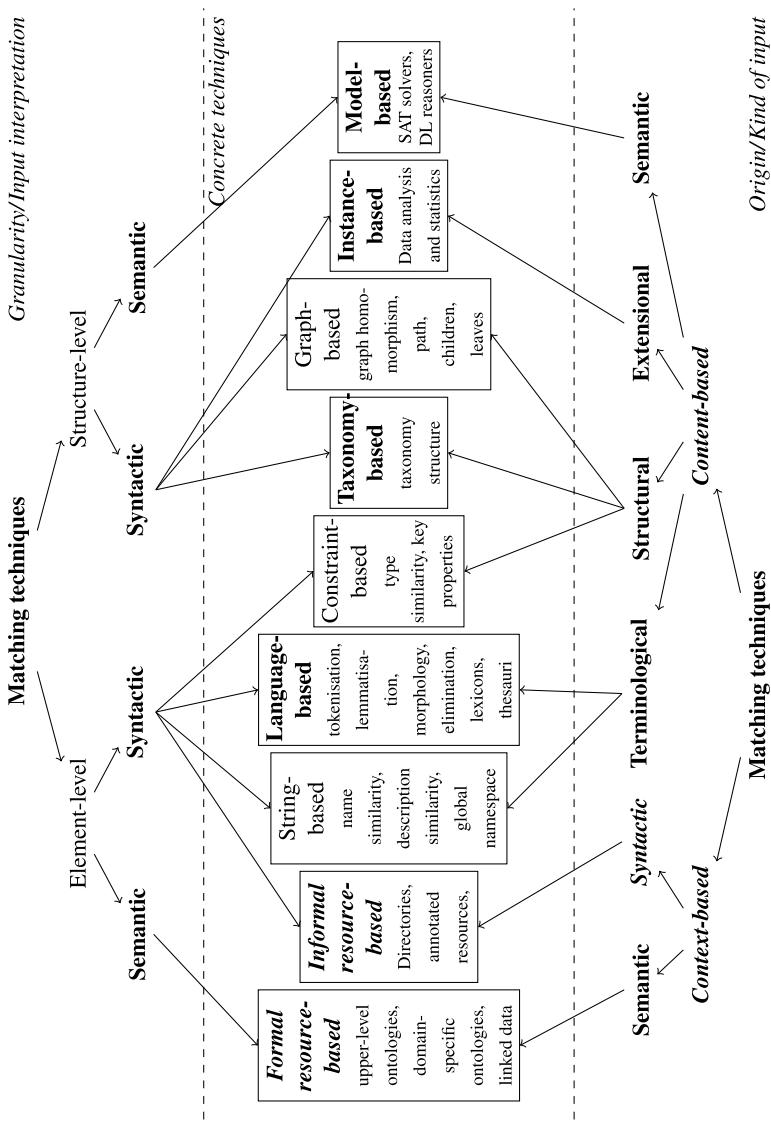


Fig. 4.1 The retained classification of matching approaches. The upper classification is based on granularity and input interpretation; the lower classification is based on the origin of information and the kind of input. The middle layer features classes of concrete matching techniques. In **bold** are new categories with respect to (Rahm and Bernstein 2001) and in **bold-italics** those introduced in the second edition.

ignoring their relations with other entities or their instances. Structure-level techniques compute correspondences by analysing how entities or their instances appear together in a structure. This criterion for schema-based approaches is the same as first introduced in (Rahm and Bernstein 2001), while the element-level vs. structure-level separation for instance-based approaches follows the work in (Kang and Naughton 2003).

- *Syntactic vs. semantic*: The key characteristic of the syntactic techniques is that they interpret the input with regard to its sole structure following some clearly stated algorithm. Semantic techniques use some formal semantics, e.g., model-theoretic semantics, to interpret the input and justify their results. In case of a semantic-based matching system, exact algorithms are complete with regard to the semantics, i.e., they guarantee a discovery of all the possible alignments, while approximate algorithms tend to be incomplete.

To emphasise the differences with the initial classification of (Rahm and Bernstein 2001), the new categories or classes are marked in bold. In particular, in the Granularity/Input Interpretation layer we detail further the element- and structure-level matching by introducing the syntactic vs. semantic distinction.

4.2.3 Origin/Kind of Input Layer

The *Origin/Kind of input* takes the origin dimension as its first level of separation, and the type of input considered by a particular technique as the second level:

- The first level is simply the content-based vs. context-based or internal vs. external distinction of the *Origin* dimension (Sect. 4.1.1).
- The second level refines these categories by distinguishing among external resources, those which are interpreted semantically, and those which are not, named *syntactic*. The *content-based matching* category is further articulated depending on which kind of data the algorithms work on strings (*terminological*), structures (*structural*), models (*semantics*) or data instances (*extensional*). The first two are found in the ontology descriptions. The third one requires some semantic interpretation of the ontology and usually uses some semantically compliant reasoner to deduce correspondences. The last one works on the actual population of an ontology. In turn, *context-based matching* is only further articulated into *syntactic* and *semantic* categories. Syntactic techniques, when considered elementary, are usually either *terminological* or *structural* or *extensional*; hence, the *informal resource-based* approaches could have been split in this three-fold way as well. However, such techniques have not been widely applied in practice so far, so we kept the presentation simpler, thus having only the *syntactic* category.

Hence, it can be considered that the *Kind of input* classification of the first edition was reduced to its first layer and that the content-based vs. context-based separation of the origin dimension was introduced on top. This follows the development of context-based matching in recent years.

4.3 Classes of Concrete Techniques

The distinctions between matching techniques in the *Concrete techniques* layer of our classification are motivated by the way in which techniques interpret the input information in each concrete case. In particular, a label can be interpreted as a string (a sequence of letters from an alphabet) or as a word or a phrase in some natural language, a hierarchy can be considered as a graph (a set of nodes related by edges) or a taxonomy (a set of concepts having a set-theoretic interpretation organised by a relation which preserves inclusion). Thus, we introduce the following classes of ontology matching techniques at the element-level: string-based, language-based, constraint-based. We also identify techniques relying on external resources related in one way or another to the ontologies to be matched. These techniques can be based on informal resources, such as text or media corpora, or formal resources, such as ontologies. At the structure-level we distinguish between graph-based, taxonomy-based, model-based, and instance-based techniques.

We discuss below the main classes of the *Concrete techniques* layer according to the above classification in more detail. Contrary to the first edition, all these classes have instances, so none of these classes are hypothetical (the hypothetical classes of the first edition have already been realised in practice). Finally, several changes have been made in this layer due to the narrowness or low representativity of some classes, namely *Alignment reuse* and *Repository of structures* are not explicitly present in Fig. 4.1 and should be considered as merged into *Formal resource-based* and *Graph-based* classes, respectively.

4.3.1 Element-Level Techniques

Element-level techniques consider ontology entities or their instances in isolation from their relations with other entities or their instances.

String-Based Techniques

String-based techniques are often used in order to match names and name descriptions of ontology entities. These techniques consider strings as sequences of letters in an alphabet. They are typically based on the following intuition: the more similar the strings, the more likely they are to denote the same concepts. Usually, distance functions map a pair of strings to a real number, such that a smaller value indicates a greater similarity between the strings. Some examples of string-based techniques that are extensively used in matching systems are prefix, suffix, edit distances, and n -gram similarity. Various such string comparison techniques are presented in Sect. 5.2.1.

Language-Based Techniques

Language-based techniques consider names as words in some natural language, e.g., English. They are based on natural language processing techniques exploiting morphological properties of the input words. Several of these techniques are presented in Sect. 5.2.2 (intrinsic techniques).

Usually, they are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results. However, we consider these language-based techniques as a separate class of matching techniques since they can be naturally extended, for example, in a distance computation (by comparing the resulting strings or sets of strings).

This class now encompasses the *Linguistic resources* class of the first edition, which covered the use of linguistic resources, such as lexicons or domain-specific thesauri, to match words (in this case names of ontology entities are considered as words of a natural language) based on linguistic relations between them, e.g., synonyms, hyponyms. Several such methods are presented in Sect. 5.2.2 (extrinsic techniques). Resources, such as thesauri and lexicons, may also be used as ‘ontologies’ instead of linguistic resources, i.e., used to interpret concepts instead of terms. In this case, these same resources occur in *Informal resource-based techniques*.

Constraint-Based Techniques

Constraint-based techniques are algorithms that deal with the internal constraints being applied to the definitions of entities, such as types, cardinality (or multiplicity) of attributes, and keys. These techniques are presented in Sect. 5.3.

Informal Resource-Based Techniques

Ontologies may be tied to informal resources, e.g., annotating encyclopedia pages or pictures. *Informal resource-based techniques* cover techniques used for deducing relations between ontology entities based on how these are related to such resources. Typically, two classes annotating the same set of pictures can be considered equivalent. Such techniques often exploit data analysis and statistical approaches as well as approaches which take advantage of a (hopefully large) corpus of related entities to find regularities and discrepancies between them. This class of techniques covers part of the *Data analysis and statistics techniques* class of the first edition dealing with extensions (instance-based matching).

Formal Resource-Based Techniques

Formal resource-based techniques take advantage of external ontologies in order to perform matching. Usually, they compose alignments between the ontologies to

be matched originating from one or several external ontologies. Several context-based ontology matchers using formal resources have been proposed in recent years, by using domain-specific ontologies, upper-level ontologies, linked data and other resources. This class of techniques covers and generalises both the *Upper-level and domain-specific formal ontologies* and *Alignment reuse* classes of the first edition of this book. Both classes focussed on different aspects of formal resource-based techniques, namely on the kind of ontologies or on the type of alignments used as external resources. A general framework for dealing with external resource-based matching, and in particular with formal resources, is detailed in Sect. 7.3.

4.3.2 Structure-Level Techniques

Contrary to element-level techniques, structure-level techniques consider the ontology entities or their instances to compare their relations with other entities or their instances.

Graph-Based Techniques

Graph-based techniques are graph algorithms which consider the input ontologies (including database schemas, and taxonomies) as labelled graphs.

Usually, the similarity comparison between a pair of nodes from the two ontologies is based on the analysis of their positions within the graphs. The intuition behind this is that, if two nodes from two ontologies are similar, their neighbours must also be somehow similar. Different graph-based techniques are described in Sect. 6.1. Along with purely graph-based techniques, there are other more specific structure-based techniques, for instance, involving trees. Graph-based techniques cover the *Repository of structure* class of the first edition, which is now considered as part of pattern-based matching (Sect. 6.1.4).

Taxonomy-Based Techniques

Taxonomy-based techniques are also graph algorithms which consider only the specialisation relation. The intuition behind taxonomic techniques is that specialisation connect terms that are already similar (being interpreted as a subset or superset of each other), therefore their neighbours may be also somehow similar. This intuition can be exploited in several different ways presented in Sect. 6.1.

Model-Based Techniques

Model-based (or semantically grounded) algorithms handle the input based on its semantic interpretation, e.g., model-theoretic semantics. The intuition is that if two

entities are the same, then they share the same interpretations. Thus, they are well grounded deductive methods. Examples are propositional satisfiability and description logics reasoning techniques. They are further reviewed in Sect. 6.5.

Instance-Based Techniques

Instance-based techniques are those that compare sets of instances of classes in order to decide if these classes match or not. They can be based on simple set-theoretic reasoning or on more elaborate data analysis and statistical techniques. They help in grouping together items or computing distances between them. From data analysis techniques, we discuss distance-based classification, formal concept analysis and correspondence analysis; from statistical analysis methods we consider frequency distributions. Instance-based techniques are mostly described in Sect. 5.4. We exclude from this category learning techniques, which require a sample of the result, i.e., the alignment. These techniques are considered specifically in Sect. 7.5.

4.4 Other Classifications

There are some other classifications of matching techniques. For example, (Ehrig 2007) introduced a classification based on two orthogonal dimensions. These can be viewed as horizontal and vertical dimensions. The horizontal dimension includes three layers that are built one on top of another:

Data layer: This is the first layer. Matching between entities is performed here by comparing only data values of simple or complex data types.

Ontology layer: This is the second layer which, in turn, is further divided into four levels, following the ‘layer cake’ of (Berners-Lee et al. 2001). These are semantic nets, description logics, restrictions and rules. For example, at the level of semantic nets, ontologies are viewed as graphs with concepts and relations, and, therefore, matching is performed by comparing only these. The description logics-level brings a formal semantics account to ontologies. Matching at this level includes, for example, determining taxonomic similarity based on the number of subsumption relations separating two concepts. This level also takes into account instances of entities, therefore, for example, assessing concepts to be the same, if their instances are similar. Matching at the levels of restrictions and rules is typically based on the idea that if similar rules between entities exist, these entities can be regarded as similar.

Context layer: Finally, this layer is concerned with the practical usage of entities in the context of an application. Matching is performed here by comparing the usages of entities in ontology-based applications. One of the intuitions behind such matching methods is that similar entities are often used in similar contexts.

The vertical dimension represents specific *domain knowledge* which can be situated at any layer of the horizontal dimension. Here, the advantage of external resources

of domain-specific knowledge, e.g., Dublin Core for the bibliographic domain, is considered for assessing the similarity between entities of ontologies.

(Doan and Halevy 2005) classifies matching techniques into (i) rule-based and (ii) learning-based. Typically, rule-based techniques work with schema-level information, such as entity names, data types and structures. Some examples of rules are that two entities match if their names are similar or if they have the same number of neighbour entities. Learning-based approaches often work with instance-level information, thereby performing matching by comparing value formats and distributions of data instances underlying the entities under consideration, for example. However, learning can also be performed at schema-level and from the previous matches, e.g., as proposed in the LSD approach (Sect. 8.2.4).

(Zanobini 2006) classifies matching methods into three categories following the cognitive theory of meaning and communication between agents:

Syntactic: This category represents methods that use purely syntactic matching methods. Examples of such methods include string-based techniques and graph matching techniques.

Pragmatic: This category represents methods that rely on comparison of data instances underlying the entities under consideration in order to compute alignments. Examples of such methods include automatic classifiers, and formal concepts analysis (Sect. 5.4.1).

Conceptual: This category represents methods that work with concepts and compare their meanings in order to compute alignments. Examples of such methods include techniques exploiting external thesauri, such as WordNet (Sect. 5.2.2), in order to compare senses among the concepts under consideration.

There were also some classifications mixing the process dimension of matching together with either input dimension or output dimension. For example, (Do 2005) extends the work of (Rahm and Bernstein 2001) by adding a *reuse-oriented* category of techniques on top of schema-based vs. instance-based separation, meaning that reuse-oriented techniques can be applied at schema- and instance-level. However, these techniques can also include some input information, such as user input or alignments obtained from previous match operations.

Finally, the more the ontology matching field progresses, the wider the variety of techniques that come into use at different levels of granularity. For example, machine learning methods, which were often applied only to the instance-level information, also started being applied more widely to schema-level information. We believe that such a cross-fertilisation will gain more support in the future. Therefore, ultimately, it could be the case that any mathematical method will find appropriate uses for ontology matching.

4.5 Summary

Following the complexity of ontology definition, a variety of techniques may be used. This chapter has shown the difficulty of having a clear cut classification of

algorithms. In Sect. 4.2, we provided two such classifications based on granularity and input interpretation on the one hand and the origin and the kind of input on the other hand.

The classifications discussed in this chapter provide a common conceptual basis for organising matching techniques. They can be used for comparing (analytically) different existing ontology matching systems as well as for designing new ones, taking advantage of state-of-the-art solutions. The classifications of matching methods also provide some guidelines which help identifying families of matching approaches.

In the following three chapters we first present basic techniques (Chap. 5), which exploit local characteristics of entities, then advanced techniques (Chap. 6), which aim at considering all the characteristics of entities, thus treating them globally, and finally, strategies (Chap. 7) used to build matching systems.

Chapter 5

Basic Similarity Measures

The goal of ontology matching is to find relations between entities expressed in different ontologies. Very often, these relations are equivalence relations that are discovered through the measure of similarity between these entities. However, more elaborate methods may directly find more precise relations.

We present here some of the basic methods for assessing the similarity or the relations between ontology entities. By basic, we mean that these methods base their judgment on one particular kind of feature of these entities. Chapter 6 considers matching techniques that compare ontology entities in a global way and Chap. 7, in turn, shows how the results of these methods may be combined.

In this chapter, we first introduce basic concepts related to similarity (Sect. 5.1). Then, we consider basic methods following the ‘kind of input’ layer of the classification of Chap. 4: entity names (Sect. 5.2), internal structure (Sect. 5.3), and extension (Sect. 5.4).

5.1 Similarity, Distances and Other Measures

There are many ways to assess the similarity between two entities. The most common way amounts to defining a measure of this similarity. We present some characteristics of these measures.

Definition 5.1 (Similarity) A similarity $\sigma : o \times o \rightarrow \mathbb{R}$ is a function from a pair of entities to a real number expressing the similarity between two objects such that

$$\forall x, y \in o, \quad \sigma(x, y) \geq 0 \quad (\text{positiveness})$$

$$\forall x \in o, \forall y, z \in o, \quad \sigma(x, x) \geq \sigma(y, z) \quad (\text{maximality})$$

$$\forall x, y \in o, \quad \sigma(x, y) = \sigma(y, x) \quad (\text{symmetry})$$

The dissimilarity is a dual operation. It is defined as follows.

Definition 5.2 (Dissimilarity) Given a set o of entities, a dissimilarity $\delta : o \times o \rightarrow \mathbb{R}$ is a function from a pair of entities to a real number such that

$$\forall x, y \in o, \quad \delta(x, y) \geq 0 \quad (\text{positiveness})$$

$$\forall x \in o, \quad \delta(x, x) = 0 \quad (\text{minimality})$$

$$\forall x, y \in o, \quad \delta(x, y) = \delta(y, x) \quad (\text{symmetry})$$

Some authors consider a ‘nonsymmetric (dis)similarity’ (Tverski 1977); we then use the term nonsymmetric measure or pre-similarity. There are more constraining notions than dissimilarity, such as distances and ultrametrics.

Definition 5.3 (Distance) A distance (or metric) $\delta : o \times o \rightarrow \mathbb{R}$ is a dissimilarity function satisfying the definiteness and triangular inequality:

$$\forall x, y \in o, \quad \delta(x, y) = 0 \quad \text{if and only if } x = y \quad (\text{definiteness})$$

$$\forall x, y, z \in o, \quad \delta(x, y) + \delta(y, z) \geq \delta(x, z) \quad (\text{triangular inequality})$$

Definition 5.4 (Ultrametric) Given a set o of entities, an ultrametric is a metric such that

$$\forall x, y, z \in o, \quad \delta(x, y) \leq \max(\delta(x, z), \delta(y, z)) \quad (\text{ultrametric inequality})$$

The measures are often normalised, especially if the similarity of different kinds of entities must be compared. Reducing each value to the same scale in proportion to the size of the considered space is the common way to normalise.

Definition 5.5 (Normalised (dis)similarity) A (dis)similarity is said to be *normalised* if it ranges over the unit interval of real numbers $[0, 1]$. A normalised version of a (dis)similarity σ (respectively, δ) is denoted by $\bar{\sigma}$ (respectively, $\bar{\delta}$).

Any normalised similarity $\bar{\sigma}$ corresponds a normalised dissimilarity $\bar{\delta} = 1 - \bar{\sigma}$ and vice versa. In the remainder, we will consider mostly normalised measures and assume that a dissimilarity function between two entities returns a real number between 0 and 1.

There are two possible ways to normalise: (i) using the maximum possible value, or (ii) using the maximum actual value. Since there might not always be a maximum possible value, all examples in the following are normalised with respect to the maximum actual value.

From the above definitions, similarity and dissimilarity are complete functions that map pairs of entities to real numbers. An alternative representation for such a function on a finite set of entities is a *matrix* (see Example 5.14). Matrices have the advantage of being finite data structures that can be exchanged between programs.

5.2 Name-Based Techniques

Some terminological methods compare strings. They can be applied to the name, the label or the comments of entities in order to find those which are similar. This can be used for comparing class names or URIs.

Throughout this section, the set \mathbb{S} will represent the set of strings, i.e., the sequences of letters of any length over an alphabet \mathbb{L} : $\mathbb{S} = \mathbb{L}^*$. The empty string is denoted by ϵ , and $\forall s, t \in \mathbb{S}, s + t$ is the concatenation of the strings s and t . $|s|$ denotes the length of the string s , i.e., the number of characters it contains. $s[i]$ for $i \in [1, |s|]$ stands for the letter in position i of s .

Example 5.6 (Strings) The string ‘article’ is made of the letters a, r, t, i, c, l and e. Its length is 7 characters. ‘peer-reviewed’ and ‘ ’ are two other strings (so ‘-’ and ‘ ’ are letters in the alphabet) and their concatenation ‘peer-reviewed’+‘ ’+‘article’ provides the string ‘peer-reviewed article’ whose length is 21.

A string s is the substring of another string t , if there exist two strings s' and s'' , such that $s' + s + s'' = t$ (denoted by $s \in t$). Two strings are equal ($s = t$) if and only if $s \in t$ and $t \in s$. The number of occurrences of s in t (denoted by $s\#t$) is the number of distinct pairs s', s'' , such that $s' + s + s'' = t$.

Example 5.7 (Substrings) The string ‘peer-reviewed article’ has the string ‘review’ as a substring because ‘peer-’+‘review’+‘ed article’=‘peer-reviewed article’. The string ‘homonymous’ has three occurrences of the string ‘o’, two occurrences of the string ‘mo’ and only one occurrence of the string ‘nym’.

The main problem in comparing ontology entities on the basis of their labels occurs due to the existence of synonyms and homonyms:

Synonyms are different words used to name the same entity. For instance, Article and Paper are synonyms in some contexts;

Homonyms are words used to name different entities. For instance, peer as a noun has a sense ‘equal’ as well as another sense ‘member of the nobility’. The fact that a word can have multiple senses is also known as *Polysemy*.

Consequently, it is not possible to deduce with certainty that two entities are the same if they have the same name or that they are different because they have different names. There are more reasons than synonymy and homonymy why this could happen. In particular:

- Words from different languages, such as English, French, Italian, Spanish, German, Greek, are used to name the same entities. For instance, the word Book in English is Livre in French and книга in Russian.
- Syntactic variations of the same word often occur according to different acceptable spellings, abbreviations, use of optional prefixes or suffixes, etc. For instance, Compact disc, CD, C.D. and CD-ROM can be considered equivalent in some contexts. However, in some other contexts, CD may mean Corps diplomatique and in some others change directory.

These kinds of variations may occur within one ontology but may be even more frequent across ontologies. However, the way in which things are named remains very important in every day communication and names remain a good index of similarity or dissimilarity. Moreover, many different techniques have been designed for assessing the similarity of two terms notwithstanding the similarity or dissimilarity of the strings which denote them.

There are two main categories of methods for comparing terms depending on their consideration of character strings only (Sect. 5.2.1) or using some linguistic knowledge to interpret these strings (Sect. 5.2.2).

5.2.1 String-Based Methods

String-based methods take advantage of the structure of the string (as a sequence of letters). String-based methods will typically find classes Book and Textbook to be similar, but not classes Book and Volume.

There are many ways to compare strings depending on the way the string is viewed: for example, as an exact sequence of letters, an erroneous sequence of letters, a set of letters, a set of words. (Cohen et al. 2003b) compare various string-matching techniques, from distance like functions to token-based distance functions. We discuss the most frequently used methods.

We distinguish between (i) normalisation techniques which are used for reducing strings to be compared to a common format, (ii) substring or subsequence techniques that base similarity on the common letters between strings, (iii) edit distances that further evaluate how one string can be an erroneous version of another, (iv) statistical measures that establish the importance of a word in a string by weighting the relation between two strings and (v) path comparisons.

Normalisation

Before comparing actual strings, which have a meaning in natural language, normalisation procedures can help improve the results of subsequent comparisons. In particular:

Case normalisation converts each alphabetic character in the strings into their lower case counterpart. For example, CD becomes cd and SciFi becomes scifi.

Diacritics suppression replaces characters with diacritic signs with their most frequent replacements. For example, replacing Montréal with Montreal.

Blank normalisation normalises all blank characters, such as blank, tabulation, carriage return, or sequences of these, into a single blank character.

Link stripping normalises some links between words, such as replacing apostrophes and blank underline into dashes or blanks. For example, peer-reviewed becomes peer reviewed.

Digit suppression suppresses digits. For example, book24545-18 becomes book.

Punctuation elimination suppresses punctuation signs. For example, C.D. becomes CD.

These normalisation operations must be used with care for several reasons. In particular:

- they are often language-dependent, e.g., they work for occidental languages;
- they are order dependent: they do not guarantee to bring the same results when applied in any order;
- they can result in losing some meaningful information; for example, carbon-14 becomes carbon or sentence separation, which is very useful for parsing, is lost;
- they may reduce variations, but increase synonyms. For example, in French *livre* and *livré* are different words respectively meaning book and delivered.

String Equality

String equality returns 0 if the strings under consideration are not identical and 1 if they are identical. This can be taken as a similarity measure.

Definition 5.8 (String equality) String equality is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that $\forall x, y \in \mathbb{S}, \sigma(x, x) = 1$ and if $x \neq y, \sigma(x, y) = 0$.

It can be performed after some syntactic normalisation of the string, e.g., down-casing, encoding conversion, diacritics normalisation.

This measure does not explain how strings are different. A more immediate way of comparing two strings is the Hamming distance which counts the number of positions in which the two strings differ (Hamming 1950). We present here the version normalised by the length of the longest string.

Definition 5.9 (Hamming distance) The Hamming distance is a dissimilarity $\delta : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\delta(s, t) = \frac{\sum_{i=1}^{\min(|s|, |t|)} s[i] \neq t[i] + ||s| - |t||}{\max(|s|, |t|)}$$

Substring Test

Different variations can be obtained from the string equality, such as considering that strings are very similar when one is a substring of another:

Definition 5.10 (Substring test) Substring test is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that $\forall x, y \in \mathbb{S}$, if there exist $p, s \in \mathbb{S}$ where $x = p + y + s$ or $y = p + x + s$, then $\sigma(x, y) = 1$, otherwise $\sigma(x, y) = 0$.

This is obviously a similarity. This measure can be refined in a substring similarity which measures the ratio of the common subpart between two strings.

Definition 5.11 (Substring similarity) Substring similarity is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that $\forall x, y \in \mathbb{S}$, and let t be the longest common substring of x and y :

$$\sigma(x, y) = \frac{2|t|}{|x| + |y|}$$

This measure is indeed a similarity. One could also consider a subsequence similarity as well. This definition can be used for building functions based on the **longest common prefix or longest common suffix**.

Thus, for example, the similarity between article and aricle would be $8/13 = .61$, while between article and paper would be $1/12 = .08$, and, finally, between article and particle would be $14/15 = .93$.

A prefix or suffix pre-similarity can be defined on this model from the prefix and suffix tests, which test whether one string is the prefix or suffix of another. These measures would not be symmetric. Prefix and suffix pre-similarity may be useful as a test for strings denoting a more general concept than another (in many languages, adding clauses to a term would restrict its range). For instance, reviewed article is more specific than article. It can also be used for comparing strings and similar abbreviations, e.g., ord and order.

The n -gram similarity is also often used in comparing strings. It computes the number of common n -grams, i.e., strings of n characters, between them. For instance, trigrams for the string article are art, rti, tic, icl, cle.

Definition 5.12 (n -Gram Similarity) Let $ngram(s, n)$ be the set of substrings of s of length n . The n -gram similarity is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}$ such that

$$\sigma(s, t) = |ngram(s, n) \cap ngram(t, n)|$$

The normalised version of this function is

$$\overline{\sigma}(s, t) = \frac{|ngram(s, n) \cap ngram(t, n)|}{\min(|s|, |t|) - n + 1}$$

This function is quite efficient when only some characters are missing.

Thus, for example, the similarity between article and aricle would be $2/4 = .5$, while between article and paper it would be 0, and, finally, between article and particle it would be $5/6 = .83$.

It is possible to add extra characters, at the beginning and end of strings, for dealing with too small strings.

Edit Distance

Intuitively, an edit distance between two objects is the minimal cost of operations to be applied to one of the objects in order to obtain the other one. Edit distances were designed for measuring a similarity between strings that may contain spelling mistakes.

Definition 5.13 (Edit Distance) Given a set Op of string operations ($op : \mathbb{S} \rightarrow \mathbb{S}$), and a cost function $w : Op \rightarrow \mathbb{R}$, such that for any pair of strings there exists a sequence of operations which transforms the first one into the second one (and vice versa), the edit distance is a dissimilarity $\delta : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ where $\delta(s, t)$, is the cost of the less costly sequence of operations which transforms s into t .

$$\delta(s, t) = \min_{(op_i)_I; op_n(\dots op_1(s))=t} \left(\sum_{i \in I} w_{op_i} \right)$$

Usually considered operations, in string edit distance, include insertion of a character $ins(c, i)$, replacement of a character by another $sub(c, c', i)$ and deletion of a character $del(c, i)$. It can be easily checked that these operations are such that $ins(c, i) = del(c, i)^{-1}$ and $sub(c, c', i) = sub(c', c, i)^{-1}$. Each operation is assigned a cost and the distance between two strings is the sum of the cost of each operation on the less costly set of operations.

The Levenshtein distance (Levenshtein 1965) is the minimum number of *insertions*, *deletions*, and *substitutions* of characters required to transform one string into the other. It is the edit distance with all costs equal to 1. The Damerau–Levenshtein distance (Damerau 1964) uses in addition the transposition operation, swapping two adjacent letters, with the same weight as the other operations. The Needleman–Wunch distance (Needleman and Wunsch 1970), in turn, is the edit distance with a higher costs for *ins* and *del*.

The edit distance is indeed a distance if $\forall op \in Op, w_{op} = w_{op^{-1}}$.

Example 5.14 The (rounded) Levenshtein distance table between the class labels of ontologies in Fig. 2.7 (p. 33):

	Science	Children	Book	Person	DVD	Textbook	Product	Pocket	Publisher	Popular	CD
Politics	0.75	1.00	0.88	0.88	1.00	1.00	0.75	0.75	0.67	0.75	1.00
Thing	0.71	0.75	1.00	1.00	1.00	0.88	1.00	1.00	0.89	1.00	1.00
Autobiography	0.92	0.85	0.85	0.92	1.00	0.85	0.92	0.92	0.85	0.85	1.00
Novel	0.86	0.88	0.80	1.00	1.00	1.00	0.86	0.67	0.89	0.71	1.00
Biography	1.00	0.89	0.78	0.89	1.00	1.00	0.89	0.89	1.00	0.89	1.00
Writer	0.86	0.75	1.00	1.00	1.00	0.88	0.86	0.83	0.67	0.86	1.00
Essay	1.00	1.00	1.00	0.83	1.00	1.00	1.00	1.00	0.89	0.86	1.00
Volume	0.86	0.75	0.83	1.00	1.00	1.00	0.71	0.83	0.78	0.71	1.00
LiteraryCritic	0.93	0.93	1.00	0.86	1.00	0.93	0.86	0.93	0.93	0.86	0.93
Poetry	0.86	0.88	0.83	0.83	1.00	0.88	0.71	0.67	0.89	0.71	1.00
Literature	0.80	0.90	1.00	0.80	1.00	0.90	0.80	0.90	0.90	0.80	1.00
Human	0.86	0.88	1.00	0.83	1.00	1.00	1.00	1.00	0.89	0.71	1.00

The closest names are Pocket and Novel, Pocket and Poetry, as well as Writer and Publisher and Politics and Publisher. These names are relatively far from each others (.67). So, in this case no correspondence can be found from such measures alone. However, the same measure on properties will obviously find the correspondence between author and author, for instance.

Other measures compute the cost of an edition operation as a function of the characters or substrings on which the operation applies. For that purpose, they use a cost matrix for each operation. A well known example of such a measure is the Smith–Waterman measure (Smith and Waterman 1981) which was adapted to compute the distance between biological sequences based on the molecules that are manipulated. Other such measures are the Gotoh (Gotoh 1981) and Monge–Elkan (Monge and Elkan 1997) distance functions.

The Jaro measure has been defined for matching proper names that may contain similar spelling mistakes (Jaro 1976, 1989). It is not based on an edit distance model, but on the number and proximity of the common characters between two strings. This measure is not a similarity because it is not symmetric.

Definition 5.15 (Jaro measure) The Jaro measure is a nonsymmetric measure $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\sigma(s, t) = \frac{1}{3} \times \left(\frac{|com(s, t)|}{|s|} + \frac{|com(t, s)|}{|t|} + \frac{|com(s, t)| - |transp(s, t)|}{|com(s, t)|} \right),$$

with $s[i] \in com(s, t)$ if and only if $\exists j \in [i - \min(|s|, |t|)/2, i + \min(|s|, |t|)/2]$ and $transp(s, t)$ are the elements of $com(s, t)$ which occur in a different order in s and t .

For instance, if we again compare article with aricle, aritcle and paper, the number of common letters will respectively be 6, 7 and 1 (because in the last case, the ‘e’ in paper is too far away from that in article). The number of transposed common letters will be 0, 1 and 0 respectively. As a consequence, the similarities between these strings are: .95, .90 and .45.

This measure has been improved by favouring matches between strings with longer common prefixes (Winkler 1999).

Definition 5.16 (Jaro–Winkler measure) The Jaro–Winkler measure $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ is as follows:

$$\sigma(s, t) = \sigma_{Jaro}(s, t) + |pref(s, t)| \times Q \times \frac{(1 - \sigma_{Jaro}(s, t))}{10},$$

such that $pref(s, t)$ is the longest prefix common to s and t , and Q is a constant.

In this case, the similarity for the three strings compared to article with $Q = 4$ are: .99, .98 and .45. These measures only improve on the previous ones by explicitly providing a model of mistakes that penalises less the comparison.

Another similar measure is SMOA (Stoilos et al. 2005) which is adapted to the way computer users define identifiers. It depends on common substring lengths and different substring lengths, the second part being subtracted from the first one. This measure has a value between -1 and 1 .

Token-Based Distances

The following techniques come from information retrieval and consider a string as a multiset of words (also called bag of words), i.e., a set in which a particular word can appear several times. These approaches usually work well on long texts (comprising many words). For that reason, it is helpful to take advantage of other strings that are attached to ontology entities. This can be adapted to ontology entities as follows:

- By aggregating different sources of strings: identifiers, labels, comments, documentation, etc. Some systems go further by aggregating the tokens that correspond to connected entities (Qu et al. 2006).
- By splitting strings into independent tokens. For example, InProceedings becomes In and Proceedings, peer-reviewed article becomes peer, reviewed and article.

Ontology entities are then identified with *bags of words* (or multisets) suitable for manipulation by using information retrieval techniques. Many different similarities or dissimilarities being applied to sets of entities can thus be applied to these bags of words. For example, the matching coefficient is the complement of the Hamming distance on sets (Sect. 5.4.1) and the Dice coefficient is the complement of the Hamming distance on multisets, i.e., using the union, intersection and cardinality of multisets instead of sets.

Original measures are those based on the corpus of such strings, i.e., the set of all such strings found in one of the ontologies or in both of them. These measures are no longer intrinsic to the strings to be compared but depend on the corpus.

They usually consider a bag of words s as a vector \vec{s} belonging to a metric space V in which each dimension is a term (or token) and each position in the vector is the number of occurrences of the token in the corresponding bag of words. This is one way to represent multisets. Each document can be considered as a point in this space identified by its coordinate vector (Salton 1971; Salton and McGill 1983).

Once the entities have been transformed into vectors, usual metric space distances can be used: Euclidean distance, Manhattan distance (also known as city blocks) and any instance of the Minkowski distance (see also p. 162). We present here the cosine similarity which measures the cosine of the angles made by two vectors. It is very often used in information retrieval.

Definition 5.17 (Cosine similarity) Given \vec{s} and \vec{t} , the vectors corresponding to two strings s and t in a vector space V , the cosine similarity is the function $\sigma_V : V \times V \rightarrow [0, 1]$ such that

$$\sigma_V(s, t) = \frac{\sum_{i \in |V|} \vec{s}_i \times \vec{t}_i}{\sqrt{\sum_{i \in |V|} \vec{s}_i^2 \times \sum_{i \in |V|} \vec{t}_i^2}}$$

Some more elaborate techniques use reduced spaces, like those obtained by correspondence analysis, in order to deal with a smaller dimension as well as to automatically map words of similar meanings to the same dimension. A famous example of such a technique, which uses singular value decomposition, is known as latent semantic indexing (Deerwester et al. 1990).

A very common measure is TFIDF (Term frequency-Inverse document frequency) (Robertson and Spärck Jones 1976) which is used for scoring the relevance of a document, i.e., a bag of words, to a term by taking into account the frequency of appearance of the term in the corpus. It is usually not a measure of similarity: it assesses the relevance of a term to a document. It is used here to assess the relevance of a substring to a string by comparing the frequency of appearance of the string in the document with regard to its frequency in the whole corpus.

Definition 5.18 (Term frequency-Inverse document frequency) Given a corpus C of multisets, we define the following measures:

$$\forall t \in \mathbb{S}, \forall s \in C, \quad tf(t, s) = t\#s \quad (\text{term frequency})$$

$$\forall t \in \mathbb{S}, \quad idf(t) = \log\left(\frac{|C|}{|\{s \in C; t \in s\}|}\right) \quad (\text{inverse document frequency})$$

$$TFIDF(s, t) = tf(t, s) \times idf(t) \quad (\text{TFIDF})$$

Many systems use measures based on TFIDF. These measures compute, for each term in the strings, its relevance with regard to the corpus based on TFIDF. Then, they use vector space techniques for computing a distance between the two strings. There are several options for doing so depending on the selected space: this can be the whole corpus, the union of terms covered by the two strings or only the intersection of the terms involved in both strings. The most often used aggregation measure is the cosine similarity.

A related type of method can be used when it is possible to establish the relevance of a particular document to some topic (with other methods) usually expressed as a probability distribution. The Kullback–Leiber divergence measure is the divergence between two probability distributions (Kullback and Leibler 1951):

Definition 5.19 (Kullback–Leiber divergence) Given a set of documents D with a probability distribution π over a set of topics T , the Kullback–Leiber divergence between two documents e and e' is as follows:

$$\delta(e, e') = \sum_{t \in T} \pi(t|e) \times \log_2\left(\frac{\pi(t|e)}{\pi(t|e')}\right)$$

This measure, which is sometimes used as a distance, is in fact not symmetric.

Path Comparison

Path difference consists of comparing not only the labels of objects but the sequence of labels of entities to which those bearing the label are related. For instance, in the left-hand ontology of Fig. 2.7, the Science class can be identified by the path Product:Book:Science. In a first approximation, these can be considered as a particular way to aggregate tokens in an ordered fashion. A simple (and only) example is the one which concatenates all the names of the superclasses of classes before comparing them. So, the result is dependent on the individual string comparison aggregated in some way.

Definition 5.20 (Path distance) Given two sequences of strings, $\langle s_i \rangle_{i=1}^n$ and $\langle s'_j \rangle_{j=1}^m$, their path distance is defined as

$$\delta(\langle s_i \rangle_{i=1}^n, \langle s'_j \rangle_{j=1}^m) = \lambda \times \delta'(s_n, s'_m) + (1 - \lambda) \times \delta(\langle s_i \rangle_{i=1}^{n-1}, \langle s'_j \rangle_{j=1}^{m-1})$$

such that

$$\delta(\langle \rangle, \langle s'_j \rangle_{j=1}^k) = \delta(\langle s_i \rangle_{i=1}^k, \langle \rangle) = k$$

with δ' being one of the other string or language-based distance and $\lambda \in [0, 1]$.

For instance, we can take the string equality distance as δ' , scoring 0 when the strings are equal, and .7 as λ . Then, if we have to compare Product:Book:Science with Book:Essay:Science and Product:Cultural:Book:Science, the distances will respectively be .273 and .09.

This measure is dependent on the similarity between the last element of each path: this similarity is affected by a λ penalty but every subsequent step is affected by a $\lambda \times (1 - \lambda)^n$ penalty. So, this measure takes into account the prefix, but the prefix can only influence the result to an extent which decreases as its distance from the end of the sequence increases. As can be seen, this measure is dependent on the rank of the elements to compare in the path. A more accurate, but expensive, measure, would choose the best match between both paths and penalise the items remote from the end of the path. Another way to take these paths into account is simply to apply them as a distance on sequences, such as described in Valtchev (1999).

Summary on String-Based Methods

The results given so far for string comparisons are useful if people use very similar strings to denote the same concepts. If synonyms with different structures are used, this will yield a low similarity. Selecting pairs of strings with low similarity, in turn, yields many false positives since two strings can be very similar, e.g., Inproceedings and proceedings, and denote relatively different concepts. These measures

Table 5.1 String measures available in SimMetrics, SecondString, OntoSim and SimPack Java packages.

SimMetrics	SecondString	OntoSim	SimPack
	<i>n</i> -grams	<i>n</i> -grams	
Levenshtein	Levenshtein	Levenshtein	Levenshtein
Jaro	Jaro	Jaro	
Jaro–Winkler	Jaro–Winkler	Jaro–Winkler	
Needleman–Wunch	Needleman–Wunch	Needleman–Wunch	Smoa
Smith–Waterman			
Monge–Elkan	Monge–Elkan		
Gotoh			
Matching coefficient			
Jaccard	Jaccard	Jaccard	Jaccard
Dice coefficient			Dice coefficient
	TFIDF	TFIDF	TFIDF
Cityblocks			Cityblocks
Euclidean			Euclidean
Cosine		Cosine	Cosine
Overlap			Overlap
Soundex			

are most often used in order to detect if two very similar strings are used. Otherwise, matching must use more reliable sources of information.

There are several software packages for computing string distances. Table 5.1 provides a brief comparison of distances available in four Java packages: SimMetrics,¹ SecondString,² OntoSim³ and SimPack.⁴ A comparison of the metrics of the second package has been provided in (Cohen et al. 2003b).

5.2.2 Language-Based Methods

So far, we have considered strings as sequences of characters. When considering language phenomena, these strings become texts (theoretical peer-reviewed journal article). Texts can be segmented into words: easily identified sequence of letters that are

¹<http://www.dcs.shef.ac.uk/~sam/stringmetrics.html>.

²<http://secondstring.sourceforge.net>.

³<http://ontosim.gforge.inria.fr>.

⁴<http://www.ifi.unizh.ch/ddis/simpack.html>.

derived from an entry in a dictionary (theoretical, peer, reviewed, journal, article). These words do not occur in a bag (as used in information retrieval) but in a sequence that has a grammatical structure. Very often words, like peer, bear a meaning and correspond to some concepts, but the more useful concepts to be properly handled in a text are terms, such as peer-review, or peer-reviewed journal.

Terms are phrases that identify concepts; they are thus often used for labelling concepts in ontologies. As a consequence, ontology matching could take great advantage of recognising and identifying them in strings. This amounts to recognise the term Peer-reviewed journal in the labels scientific periodicals reviewed by peers (and not in journal review paper).

Language-based methods rely on using Natural Language Processing (NLP) techniques to help extract the meaningful terms from a text. Comparing these terms and their relations should help assess the similarity of the ontology entities they name and comment. Although these are based on some linguistic knowledge, we distinguish methods which rely on algorithms only and those which make use of external resources such as dictionaries.

Intrinsic Methods: Linguistic Normalisation

Linguistic normalisation aims at reducing each form of a term to some standardised form that can be easily recognised. Table 5.2 shows that the same term (theory paper) can appear under many different forms. One may distinguish three main kinds of term variation (Maynard and Ananiadou 2001): morphological (variation on the form and function of a word based on the same root), syntactic (variation on the grammatical structure of a term) and semantic (variation on one aspect of the term, usually using a hypernym or hyponym). Multilingual variation, i.e., such that the term variant is expressed in a different language, can be naturally added to these. Various subtypes of these broad categories are exemplified in Table 5.2. Moreover, these types of variations may be combined in various ways.

Complete linguistic software chains have been developed for quickly obtaining a normal form of strings denoting terms. This is available through shallow parsers or part-of-speech taggers (Brill 1992). Lucene⁵ (McCandless et al. 2010) is an open framework, which includes a lemmatiser (SnowBall). Gate⁶ (Cunningham et al. 2011) is a linguistic workbench that covers all of the following functions:

Tokenisation: Tokenisation is the operation described in Sect. 5.2.1. It consists of segmenting strings into sequences of tokens through recognising punctuation, cases, blank characters, digits, etc. For example, peer-reviewed periodic publication becomes ⟨peer, reviewed, periodic, publication⟩.

Lemmatisation: The strings underlying tokens are morphologically analysed in order to reduce them to normalised basic forms. Morphological analysis makes

⁵<http://lucene.apache.org>.

⁶<http://gate.ac.uk>.

Table 5.2 Variants of the term *theory paper* (adapted from (Maynard 1999) and (Euzenat et al. 2004)).

Type	Subtype	Example
Morphological	Inflection	theory papers
	Derivation	theoretical paper
	Inflectional-Derivational	theoretical papers
Syntactic	Insertion	theory review paper
	Permutation	paper on theory
	Coordination	philosophy and theory paper
Morphosyntactic	Derivation-Coordination	philosophical and theoretical paper
	Inflection-Permutation	papers on theory
Semantic		foundational paper
Multilingual	French	article théorique

it possible to find flexions and derivations of a root. This involves suppressing tense, gender or number marks. Retrieving the root is called lemmatisation. Systems can use some approximate lemmatisation techniques called stemming (Lovins 1968; Porter 1980) which strip suffixes from terms. For example, reviewed becomes review.

Term extraction: More elaborate technologies enable the extraction of terms from a text (Jacquemin and Tzoukermann 1999; Bourigault and Jacquemin 1999; Maynard and Ananiadou 2001; Cerbah and Euzenat 2001). It is generally related to what is called corpus linguistics and requires a relatively large amount of text. Terminology extractors identify terms from the repetition of morphologically similar phrases in the texts and the use of patterns, e.g., noun¹ noun² → noun² on noun¹. This would recognise that the term theory paper is the same term as paper on theory.

Stopword elimination: Those tokens that are recognised as articles, prepositions, conjunctions, etc. (usually words, such as to or a), are marked to be discarded because they are considered as meaningless (empty) words for matching. For example, collection of article becomes collection article.

Once these techniques have been applied, ontology entities are represented as sets of terms, not words, that can be compared with the same techniques as presented before.

Extrinsic Methods

Extrinsic linguistic methods use external resources, such as dictionaries and lexicons. Several kinds of linguistic resources can be exploited in order to find similarities between terms.

Lexicons. A lexicon, or dictionary, is a set of words together with a natural language definition of these words (see for instance those of Example 5.22). For a particular word, e.g., Article, there can be several such definitions. Dictionaries can be used with gloss-based distances (see below).

Semantico-syntactic lexicons. Semantico-syntactic lexicons and semantic lexicons are resources used in natural language analysers. They very often not only record names but their categories, e.g., inanimate, liquid, and record the types of arguments taken by verbs and adjectives, e.g., to flow takes a liquid as subject and has no object. These are difficult to create and are not much used in ontology matching.

Thesauri. A thesaurus is a kind of lexicon to which some relational information has been added. It usually contains relations, named hypernym, e.g., Biography is a more general term, than Autobiography, which is hyponym, synonym, e.g., Paper means the same as Article, antonym, e.g., practice is the opposite of theory. WordNet (Miller 1995) is such a thesaurus that distinguishes clearly between word senses by grouping words into sets of synonyms (synsets).

Terminologies. A terminology is a thesaurus for terms, which very often contains phrases rather than single words. They are usually domain-specific and tend to be less equivocal than dictionaries.

This is not an exhaustive nor an authorised description of linguistic resources but it provides a typology of the kinds of properties on which a similarity between terms can be assessed on a linguistic basis.

These resources may be defined for one language or be specific to some domain. In the latter case, they tend to be more adapted when texts or ontologies concern this domain because they retain specialised senses, or senses that do not exist in the everyday language. They may also contain proper names and common abbreviations that are used in the domain. For instance, a company could expand CD as Compact Disc, or PO as Purchase Order instead of Post Office or Project Officer.

Linguistic resources are introduced in order to deal with synonyms (since matching entities are named differently). By increasing the interpretation (sense) of words, they increase the chances of finding the matching terms (true positives). However, this also increases homonyms (the fact that more words are available for naming the matching entities) and the chances of incorrectly matching nonmatching terms (false positives). Dealing with this problem is known as word sense disambiguation (Lesk 1986; Ide and Véronis 1998). It tries to restrict the candidate senses (and the candidate matches) from the context, especially by selecting the senses in relation to the other associated words and their senses. Word-sense disambiguation techniques have been intensively used in ontology matching (Gracia 2009). Some systems, such as Blooms (Sect. 8.1.34), use the disambiguation pages of Wikipedia as a way to disambiguate terms used in ontologies.

We illustrate the use of external resources with the help of WordNet⁷ (Miller 1995; Fellbaum 1998). WordNet is an electronic lexical database for English (it has

⁷<http://wordnet.princeton.edu>.

been adapted to other languages; see for instance EuroWordNet⁸), based on the notion of *synsets* or sets of synonyms. A synset denotes a concept or a sense of a group of terms. WordNet also provides an hypernym (superconcept/subconcept) structure as well as other relations such as meronym (*part of* relations). It also provides textual descriptions of the concepts (*gloss*) containing definitions and examples. We will consider WordNet as a partially ordered synonym resource.

Definition 5.21 (Partially ordered synonym resource) A partially ordered synonym resource Σ over a set of words W , is a triple $\langle E, \leq, \lambda \rangle$, such that $E \subseteq 2^W$ is a set of synsets, \leq is the hypernym relation between synsets and λ is a function from synsets to their definition (a text that is considered here as a bag of words in W). For a term t , $\Sigma(t)$ denotes the set of synsets associated with t .

Example 5.22 (WordNet entry) We reproduce here the WordNet (version 2.0) entry for the word author. Each sense is numbered in superscript:

author¹ *noun*: Someone who originates or causes or initiates something; *Example* ‘he was the generator of several complaints’. *Synonym* generator, source. *Hypernym* maker. *Hyponym* coiner.

author² *noun*: Writes (books or stories or articles or the like) professionally (for pay). *Synonym* writer². *Hypernym* communicator. *Hyponym* abstractor, alliterator, authoress, biographer, coauthor, commentator, contributor, cyberpunk, drafter, dramatist, encyclopedist, essayist, folk writer, framer, gagman, ghostwriter, Gothic romancer, hack, journalist, librettist, lyricist, novelist, pamphleteer, paragrapher, poet, polemist, rhymer, scriptwriter, space writer, speechwriter, tragedian, wordmonger, word-painter, wordsmith, Andersen, Assimov...

author³ *verb*.: Be the author of; *Example* ‘She authored this play’. *Hypernym* write. *Hyponym* co-author, ghost.

This resembles a traditional dictionary entry apart from the *Hypernym* and *Hyponym* features and the explicit mention of the considered sense. The hypernym relations for the senses of the words creator, writer, author, illustrator, and person are presented in Fig. 5.1.

There are at least three families of methods for using WordNet as a resource for matching terms used in ontology entities:

- considering that two terms are similar because they belong to some common synset;
- taking advantage of the hypernym structure for measuring the distances between synsets corresponding to two terms;
- taking advantage of the definitions of concepts provided by WordNet in order to evaluate the distance between the synsets associated with two terms.

⁸<http://www illc uva nl/EuroWordNet/>.

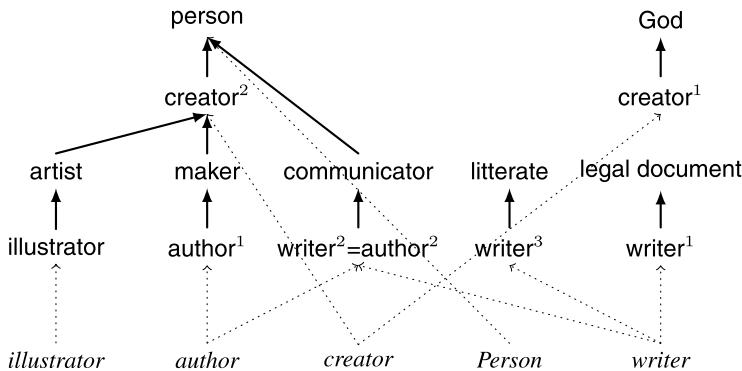


Fig. 5.1 The fragment of the WordNet hierarchy (limited to nouns) dealing with author, writer, creator, illustrator and person.

A matcher based on WordNet can be designed by translating the (lexical) relations provided by WordNet to logical relations according to the following rules (Giunchiglia et al. 2004):

- $t \leq t'$, if t is a hyponym or meronym of t' . For example, author is a hyponym of creator, therefore we conclude that $\text{author} \sqsubseteq \text{creator}$.
 - $t \geq t'$, if t is a hypernym or holonym of t' . For example, Europe is a holonym of France, therefore we conclude that $\text{Europe} \sqsupseteq \text{France}$.
 - $t = t'$, if they are connected by synonymy relation or they belong to one synset. For example, writer and author are synonyms, therefore we conclude that $\text{writer} = \text{author}$.
 - $t \perp t'$, if they are connected by antonymy relation or they are the siblings in the *part of* hierarchy. For example, Italy and France are siblings in the WordNet *part of* hierarchy, therefore we conclude that $\text{Italy} \perp \text{France}$.

Simple measures may be defined here (we only consider synonyms because they are the basis of WordNet synsets but other relationships may be used as well). The simplest use of synonyms is as follows:

Definition 5.23 (Synonymy similarity) Given two terms s and t and a synonym resource Σ , the synonymy is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\sigma(s, t) = \begin{cases} 1 & \text{if } \Sigma(s) \cap \Sigma(t) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

This would consider that the similarity between author and writer is maximal (1.) and that between author and creator is minimal (0.).

Example 5.24 (Synonymy) The synonymy similarity between illustrator, author, creator, Person, and writer is given by the following table:

	illustrator	author	creator	Person	writer
illustrator	1.	0.	0.	0.	0.
author	0.	1.	0.	0.	1.
creator	0.	0.	1.	0.	0.
Person	0.	0.	0.	1.	0.
writer	0.	1.	0.	0.	1.

This strict exploitation of synonyms does not allow analysis of how far nonsynonymous objects are nor how close synonymous objects are. Since synonymy is a relation, all the measures on the graph of relations can be used on WordNet synonyms. Another measure computes the cosynonymy similarity.

Definition 5.25 (Cosynonymy similarity) Given two terms s and t and a synonym resource Σ , the cosynonymy is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\sigma(s, t) = \frac{|\Sigma(s) \cap \Sigma(t)|}{|\Sigma(s) \cup \Sigma(t)|}$$

Example 5.26 (Cosynonymy similarity) The cosynonymy similarity between illustrator, author, creator, Person, and writer is given by the following table:

	illustrator	author	creator	Person	writer
illustrator	1.	0.	0.	0.	0.
author	0.	1.	0.	0.	.25
creator	0.	0.	1.	0.	0.
Person	0.	0.	0.	1.	0.
writer	0.	.25	0.	0.	1.

Some elaborate measures take into account that the terms may be part of several synsets and use a measure in the hyponym-hypernym hierarchy between synsets. A simple measure, known as edge-count, counts the number of edges separating two synsets in Σ (or the structural topological dissimilarity; see Sect. 6.1.1). More elaborate measures weight edge count with the position of synsets in the hierarchy, such as the one proposed by Wu and Palmer (see Sect. 6.1.1). All measures defined in Sect. 6.1.1 can be used on the WordNet hypernym graph if they can deal with unrooted directed acyclic graphs.

Other measures rely on an information-theoretic perspective. They are based on the assumption that the most probable a concept, the less information it carries. So the information content of a concept is inverse to its probability of occurrence. In the similarity proposed in (Resnik 1995, 1999), each synset (c) is associated with a probability of occurrence ($\pi(c)$) of an instance of the concept in a particular corpus. Usually, $\pi(c)$ is the sum of the synset word occurrences divided by the total number of concepts. This probability is obtained from a corpus study. It is such that the

more specific the concept, the lower its probability. The Resnik semantic similarity between two terms is a function of the more general synset common to both terms. It considers the maximum information content (or entropy), of the possible such synsets, taken as the negation of the logarithm of the probability of occurrence.

Definition 5.27 (Resnik semantic similarity) Given two terms s and t and a partially ordered synonym resource $\Sigma = \langle E, \leq, \lambda \rangle$ provided with a probability measure π , Resnik semantic similarity is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\sigma(s, t) = \max_{k; \exists c, c' \in E; s \in c \wedge t \in c' \wedge c \leq k \wedge c' \leq k} (-\log(\pi(k)))$$

We do not provide examples of corpus-based similarity because results depend on the corpus on which it is based (here for defining π). Examples of such measures based on the Brown corpus⁹ are given in (Budanitsky and Hirst 2006).

This measure uses the maximum, but one could have chosen instead an average or a sum of all the pairs of synsets associated with the two terms.

Other information-theoretic similarities depend on the increase of the information content measure from the terms to their common hypernyms instead of the shared information content. This is the case in the Jiang–Conrath method (Jiang and Conrath 1997) or the Lin information-theoretic similarity (Lin 1998). This method specifies the probabilistic degree of overlap between two synsets:

Definition 5.28 (Information-theoretic similarity) Given two terms s and t and a partially ordered synonym resource $\Sigma = \langle E, \leq, \lambda \rangle$ provided with a probability π , Lin information-theoretic similarity is a similarity $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$ such that

$$\sigma(s, t) = \max_{k; \exists c, c' \in E; s \in c \wedge t \in c' \wedge c \leq k \wedge c' \leq k} \frac{2 \times \log(\pi(k))}{\log(\pi(s)) + \log(\pi(t))}$$

These similarities are not normalised.

A final way to compare terms found in strings through a thesaurus, like WordNet, is to use the definition (gloss) given to these terms. In this case, any dictionary entry $s \in \Sigma$ is identified by the set of words corresponding to $\lambda(s)$. Then any measure defined in Sect. 5.2.1 can be used for comparing the strings (Lesk 1986).

Definition 5.29 (Gloss overlap) Given a partially ordered synonym resource $\Sigma = \langle E, \leq, \lambda \rangle$, the gloss overlap between two strings s and t is defined by the Jaccard similarity between their glosses:

$$\sigma(s, t) = \frac{|\lambda(s) \cap \lambda(t)|}{|\lambda(s) \cup \lambda(t)|}$$

⁹<http://icame.uib.no>.

Example 5.30 (Gloss overlap) For computing the gloss overlap similarity between illustrator, author, creator, Person, and writer, we used the following treatments: take gloss for all senses and add the term name; suppress quotations ('...'); suppress empty words (or, and, the, a, an, for, of, etc.); suppress technical vocabulary, e.g., ‘term’; suppress empty phrases, e.g., ‘usually including’; keep categories, e.g., law; stem words. The gloss of author is given in Example 5.22.

The results have been taken as sets (not bags, so there is no repetition) of words and syntactically compared, yielding the following table:

	illustrator	author	creator	Person	writer
illustrator	1.	0.05	0.07	0.	0.02
author	0.05	1.	0.	0.	0.19
creator	0.07	0.	1.	0.06	0.02
Person	0.	0.	0.06	1.	0.04
writer	0.02	0.19	0.02	0.04	1.

This result is consistent with the previous measures since the only previously matching pair (author-writer) is still the highest scorer. This measure introduces new relations such as creator-illustrator, but still does not find the (possible) relation between creator and author. This is entirely related to the quality of glosses in WordNet.

Another example of building a matcher by using (WordNet) glosses includes counting the number of occurrences of the label of the source input sense in the gloss of the target input sense. If this number is equal to a threshold, e.g., 1, the less general relation can be returned. The reason for returning the less general relation is due to a common pattern of defining terms in glosses through a more general term. For example, in WordNet creator is defined as ‘a person who grows or makes or invents things’. Thus, following this strategy we could find that creator \sqsubseteq person. Some other variations of gloss-based matchers include considering glosses of the parent (children) nodes of the input senses in the WordNet *is a (part of)* hierarchy (Giunchiglia and Yatskevich 2004). The relations produced by these matchers depend heavily on the context of the matching task, and therefore, these matchers cannot be applied in all the cases (Giunchiglia et al. 2006c).

Multilingual Methods

Ontologies may be monolingual if all their labels are available in only one natural language, or multilingual if they use several different languages. Moreover, due to the possibility of ontology languages, such as OWL or SKOS, to declare several labels with explicitly identified languages, it is possible to assess the language in which ontology entities are identified, e.g., Article@fr is the word ‘article’ in French.

Similarly, two terms used as labels can be matched monolingually if they belong to the same natural language, or crosslingually if they belong to two different natural languages. The first option corresponds to the techniques which were considered in

the previous sections. However, crosslingual matching is taking on more importance due to the variety of languages in which ontologies can be described. It may be performed in several ways: by comparing to a pivot language or through cross-translation (Trojahn et al. 2010b).

Comparison to a pivot language translates the terms of two ontologies into one particular language (Jung et al. 2009). This reduces the number of required translations, allowing for dealing with several languages of a multilingual ontology in one pass. Having the same entity identified by labels in different languages in one ontology may help in disambiguating the target term in the pivot language. For instance, Paper in English may be translated into Papel or Artículo in Spanish, but if the French label is Article, then the appropriate sense is Artículo. Usually, pivot language methods are chosen because linguistic resources for comparing the terms in different languages are available.

Cross-translation translates all the terms of one ontology into the language of the other ontology (Fu et al. 2012). Translation may use a variety of techniques from the use of interlingual resources, such as interlingual lexicons, i.e., lexicons in which the definition is replaced by the equivalent terms in another language, e.g., Paper in English corresponds to Article in French. Such dictionaries can be very useful if ontology labels are expressed in different languages. They can be used for matching as well as for disambiguating terms, i.e., identifying their intended sense, before matching. Translation may also use other resources, such as on-line statistical translators (Trojahn et al. 2010b).

Once all the terms are in only one language, either the pivot language or the target language in cross-translation, usual (monolingual) techniques can be used to compare them. This has consequences for linguistic matching techniques and brings the distinction between:

monolingual matching, which matches two ontologies based on their labels in a single language, such as English;

multilingual matching, which matches two ontologies based on labels in a variety of languages, e.g., English, French and Spanish. This may be achieved by parallel monolingual matching of terms or crosslingual matching of terms in different languages;

crosslingual matching, which matches two ontologies based on labels in two identified different languages, e.g., English vs. French.

These definitions are slightly different from those of Spohr et al. (2011), where the second option of multilingual matching was called crosslingual. It also makes multilingual matching more general than the two other options (which are multilingual matching with only one pair of languages).

An interesting technique offered by multilingual ontologies is crosslingual validation. It consists of matching two ontologies in parallel (Sect. 7.2) in different languages, e.g., French matched with French, English with English, and aggregating these parallel matches in order to retain a consensus (Sect. 7.4). Usually, if a particular correspondence is found by a sufficient number of matchers, then it is retained. This approach was found beneficial for matching accuracy (Spohr et al. 2011).

Table 5.3 List of language measures based on WordNet and available in the wn-similarity Perl package and the SimPack Java package (some measures have not been presented yet).

WordNet::similarity	SimPack
Resnik	Resnik
Jiang–Conrath (Jiang and Conrath 1997)	
Lin	Lin
Leacock–Chodorow	Leacock–Chodorow
Hirst–St. Onge (Saint-Onge 1995)	
Edge count	Edge count
Wu–Palmer	Wu–Palmer
Extended gloss overlap	
Vector on gloss	

Summary on Linguistic Methods

Many methods presented in this section have been implemented in the Perl package¹⁰ WordNet::similarity (Pedersen et al. 2004) and the Java package SimPack⁴ (see Table 5.3). They have been thoroughly compared in (Budanitsky and Hirst 2006).

Linguistic resources, such as stemmers, part-of-speech taggers, lexicons, and thesauri are invaluable resources since they allow the interpretation of the terms used in the expressions of ontologies. They provide a more accurate apprehension of these labels.

However, whenever the adequate resources are available for some language, they mainly open new possible matches between entities because they recognise that two terms can denote the same concept. Unfortunately, since they also recognise that the same term may denote several concepts at once, these techniques provide many possible matches from which to choose.

One way to choose among these representations is to take into account the structure of ontology entities in order to select the most coherent matches.

5.3 Internal Structure-Based Techniques

The structure of entities in ontologies can be compared, instead of or in addition to comparing their names or identifiers.

This comparison may be subdivided into a comparison of the internal structure of an entity, i.e., besides its name and annotations, its properties or, in the case of OWL ontologies, the properties which take their values in a data type, or the comparison of the entity with other entities to which it is related. The former is called internal structure and the latter is called relational structure. The internal structure is the

¹⁰<http://wn-similarity.sourceforge.net>.

definition of entities without reference to other entities; the relational structure is the set of relations that an entity has with other entities. As expected, the internal structure is primarily exploited in database schema matching, while the relational structure is more important in matching formal ontologies and semantic networks. This section is only concerned with internal structure-based techniques; external or relational techniques are considered in Sect. 6.1.

Internal structure-based methods are sometimes referred to as constraint-based approaches in the literature (Rahm and Bernstein 2001). These methods are based on the internal structure of entities and use such criteria as the set of their properties, the range of their properties (attributes and relations), their cardinality or multiplicity, and the transitivity or symmetry of their properties to calculate the similarity between them.

Entities with comparable internal structures or properties with similar domains and ranges in two ontologies may be numerous. For that reason, these kinds of methods are commonly used to create correspondence clusters rather than to discover accurate correspondences between entities. They are usually combined with other element-level techniques, such as terminological methods, and are responsible for reducing the number of candidate correspondences. They can be used with other approaches as a preprocessing step to eliminate most of the properties that are clearly incompatible.

For illustrating these methods we consider the properties associated with the Product and Volume entities in the example of Fig. 5.2 (the expected correspondences are given in Fig. 2.9, p. 45).

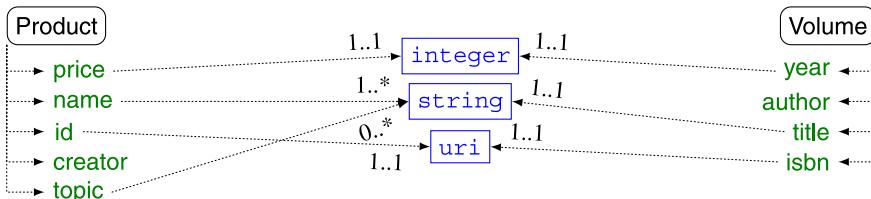


Fig. 5.2 Two sets of properties to be compared.

If we start from the elements of Fig. 5.2, there is no chance that pure terminological similarity methods find them very similar, though year and creator may appear the same to some edit distance methods. A linguistic method may be better able to find a relationship between creator and author.

Comparing the internal structure of ontology entities may be reduced to comparing their properties and composing the obtained result: the system can evaluate the similarity between all components considered next (names, keys, data types, domains, cardinalities) or multiplicities and combine the results. The combination operation is considered in Sect. 7.4, we focus here on the elementary comparison.

5.3.1 Property Comparison and Keys

In database schemas, unlike in formal ontologies, tables are provided with keys: a combination of properties whose values uniquely identify an object. For a Book, it would typically be the international standard book number (isbn), for a Person it can be his or her name, birth place and date.

This information is primarily very useful for recognising that two individuals are the same. Thus, keys are mostly used in extensional methods as a means to identify individuals and then apply methods on common set of instances (Sect. 5.4).

However keys can also be used for identifying classes: two classes identified in the same way are likely to represent the same set of objects. Moreover, even if two schemas use different keys for the same class, e.g., identifying Person with a social security number, there can be secondary keys that perform the same functions, e.g., that the social security number is also considered a key in the other class. So, when provided with keys, if they are highly compatible (similar names and types), it is plausible that the classes are equivalent.

For instance, if Product has id as a key and Volume has isbn as a key, it can be considered that these properties should correspond in case where the classes are the same. This can be considered possible because both properties have the same type (uri).

5.3.2 Data Type Comparison

Property comparison involves comparing the property data type (in OWL, this can be the range of the relation or a Restriction applied to the property in the class). Contrary to objects that require interpretations, data types can be considered objectively and it is possible to determine how close a data type is to another (ideally this can be based on the interpretation of data types as sets of values and the set-theoretic comparison of these data types (Valtchev 1999; Valtchev and Euzenat 1997)).

We distinguish here between a data type, which corresponds to the way the values are stored in a computer (like integer, float, string or uri), and a domain, which characterises a subset of a particular data type (like [10 12] or ‘*book’). Data types are considered here and domains are addressed in the next section.

Data types are not fully disjoint, though there are rules by which an object of one type can be thought of as an object of another type and rules by which a value of some type can be converted in the memory representation of another type (known as *casting* in programming languages).

Ideally, the proximity between data types should be maximal when these are the same types, lower when the types are compatible (for instance, integer and float are compatible since they can be cast one into the other) and the lowest when they are non compatible. In addition, domain comparison should ideally be based on data type comparison and the comparison of the sets of values covered by these domains. The compatibility between property data types may be assessed by using an underlying look-up table. An example of a part of such a table is given in Table 5.4.

Table 5.4 Part of a data type compatibility table.

	char	fixed	enumeration	int	number	string
string	0.7	0.4	0.7		0.4	0.5
number	0.6	0.9	0.0		0.9	1.0

Such a table can be extracted, for languages like OWL, from the type hierarchy of XML Schema data types (see Fig. 5.3). In the example of Fig. 5.2, it can be considered that since a `uri` is a subclass of `string`, the `isbn` may be related to `name`.

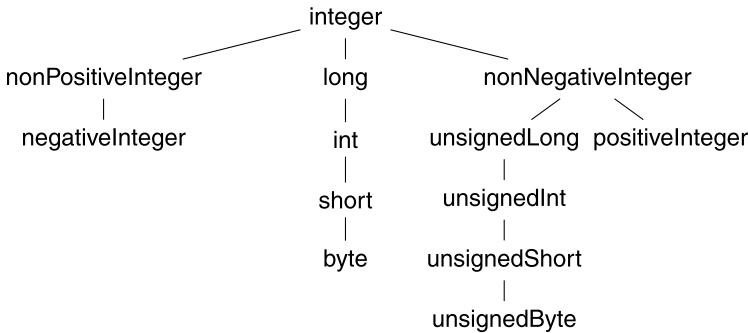


Fig. 5.3 Fragment of the XML Schema data type hierarchy (Biron and Malhotra 2004).

Example 5.31 (Data type comparison) In the example of Fig. 5.2, data type comparison would match `price` with `year`, both `name` and `topics` with `title`, and `id` with `isbn`. `creator` and `author` are left aside because they are object-valued properties. This comparison yields interesting results since it finds the expected matches. However, it also finds incorrect ones (`price-year` and `topics-title`), so these methods cannot be used in isolation.

5.3.3 Domain Comparison

Depending on the entities to be considered, what can be reached from a property may be different: in classes these are domains while in individuals these are values. Moreover, they may be structured in sets or sequences. It is thus important to consider this fact in the comparison.

Types or domains of properties may be compared on the basis of their interpretations: sets of values (Valtchev 1999). Type comparison is based on their respective size, in which the size of a type is the cardinality or multiplicity of the set of values it defines. The distance between two domains is then given by the difference between their size and that of their common generalisation. This measure is usually

normalised by the size of the largest possible distance attached to a particular data type. We give here an instance of this type of measure.

Definition 5.32 (Relative size distance) and e' over a data type τ , the relative size distance $\delta : 2^\tau \times 2^\tau \rightarrow [0, 1]$, is as follows:

$$\delta(e, e') = \frac{|gen_\tau(e \vee e')| - |gen_\tau(e \wedge e')|}{|\tau|},$$

such that $gen_\tau(\cdot)$ provides the generalisation of a type expression and \vee and \wedge correspond to the union and intersection of the types.

Example 5.33 (Relative size distance) Consider a property age in one class to be compared with the property age of three other classes (schoolchild, teenager and grown-up). The first property has a domain of [6 12], while the others have respective domains expressed by: [7 14], [14 22] and ≥ 10 . All these properties have data type integer. The generalisation of these four domains are the domains themselves, the union with [6 12] is respectively [6 14], [6 22], [6 $+\infty$ [, and the intersection is respectively [7 12], \emptyset , and [10 12]. As a consequence, the distance will be respectively $3/|\tau|$, $17/|\tau|$ and $|\tau| - 3/|\tau|$. This corresponds to some intuition that the distance between domains depends on the difference between the values they cover in isolation and in common.

There are three advantages of this measure. The most obvious one is that it is normalised. The second one is that it is general (it is not expressed in terms of integers). The third one is that it can easily be mapped to the usual measures that are often used.

Usually, a common generalisation depends on the type: it is a set for enumerated types and an interval for ordered types (it can also be a set of intervals). In the case of dense types, the size of a domain is the usual measure of its size (Euclidean distance can be used for real or floating point numbers). The case of infinite types has to be taken adequately (by evaluating the largest possible domain in a computer or by normalising with regard to the actual corpus) (Valtchev 1999). Normalising over the largest distance in the corpus, if possible, is often a good idea. Indeed, it is not reasonable, for example, to normalise the age of people with that of planets or their size even if they use the same unit. Another advantage of this framework is that it encompasses value comparisons which can be considered as singletons and compared with domains if necessary.

5.3.4 Comparing Multiplicities and Properties

Properties may be constrained by multiplicities (as they are called in UML). Multiplicities are the acceptable cardinalities of the set of values of a property (for a given object). Similar to compatibilities between data types, compatibility between

cardinalities can be established based on a table look-up. An example of such a table for DTDs is given in Table 5.5, following (Lee et al. 2002).

Table 5.5 A cardinality compatibility table.

	*	+	?	none
*	1.0	0.9	0.7	0.7
+	0.9	1.0	0.7	0.7
?	0.7	0.7	1.0	0.8
none	0.7	0.7	0.8	1.0

In OWL, cardinalities or multiplicities are expressed through the `minCardinality`, `maxCardinality` and `cardinality` restrictions. Multiplicities can be expressed as an interval of the set of positive integers $[0 +\infty[$. As such, they are domains of the integer type. Two multiplicities are compatible if the intersection of the corresponding intervals is nonempty. Any measure on the integer data type can be used for assessing the similarity between multiplicities (see Sect. 5.3.3), in this case a simpler distance inspired from the Jaccard similarity.

Values may be collected by a particular construction (set, list, multiset) on which cardinality constraints are applied. Again, it is possible to compare these constructed data types by comparing (i) the data types on which they are constructed and (ii) the cardinalities that are applied to them. For instance, sets of 2 and 3 children are closer to a set of 3 people than to a set of 10–12 flowers (if children are people). This technique is used in (Euzenat and Valtchev 2004).

Definition 5.34 (Multiplicity similarity) Given two multiplicity expressions $[b \ e]$ and $[b' \ e']$, the multiplicity similarity is a similarity between nonnegative integer intervals $\sigma : 2^\tau \times 2^\tau \rightarrow [0 \ 1]$, such that

$$\sigma([b \ e], [b' \ e']) = \begin{cases} 0 & \text{if } b' > e \text{ or } b > e' \\ \frac{\min(e, e') - \max(b, b')}{\max(e, e') - \min(b, b')} & \text{otherwise} \end{cases}$$

For instance, if we have to compare multiplicity $[0 \ 6]$ with $[2 \ 8]$, $[8 \ 12]$ and $[0 +\infty]$, the comparison will respectively yield .5, 0. and $6/MAXINT$ (the latter is very low but remains nonnull because it is compatible with the initial multiplicity).

Example 5.35 (Multiplicity comparison) In the example of Fig. 5.2, multiplicity comparison can be used to further match `id` with `isbn` because they will both have a cardinality of $[1 \ 1]$ and, unfortunately, will match `price` with `year` as well. However, it can also be used to prefer matching `name` rather than `topic` to `title` because they have the same multiplicities ($[1 +\infty]$ instead of $[0 +\infty]$).

5.3.5 Other Features

Other internal structural factors have been considered in database schema matching. Among these additional property characteristics are uniqueness, static semantic integrity constraints, dynamic semantic integrity constraints, security constraints, allowable operations and scale (Navathe and Buneman 1986). Since these are internal features, they may be very dependent on the knowledge model.

It is also possible in some languages to consider collection constructors, e.g., Set, List, Bag or multiset, Array, and their compatibility. It is then necessary to compare sets or lists of objects, e.g., the sequence of topics or the set of authors of a Book. In this case, general techniques may be used for assessing the similarity or distance between these sets depending on the similarity applying to the type of their elements. Concerning sets, these methods will be presented in Sect. 5.4.1 in the context of extension comparison. Concerning sequences, they can be adapted from those measures of Sect. 5.2.1 that consider strings as sequences of characters and paths as sequences of strings. In addition, Sect. 6.2.2 explains how to compare sets of objects with similarities.

In (Ehrig and Sure 2004), a set of rules is used for determining similarity between ontology entities. The authors point out that some features from OWL related to internal structure, such as symmetry and restrictions of values, could be used, but are discarded at the moment, as they do not have any wide distribution.

Summary on Internal Structure-Based Techniques

Internal structure, including the names of entities, is very important for matching because it provides a basis on which algorithms can rely. The techniques for comparing them are efficient and easy to implement.

However, the internal structure does not provide much information on the entities to compare: many very different types of objects can have properties with the same data types. On the one hand, they can be used for eliminating incompatible correspondences and promoting compatible ones. On the other hand, it is always possible that different models of a concept use different, and incompatible, types. For these reasons, internal structure comparisons must always be used jointly with other techniques.

5.4 Extensional Techniques

When individual representations (or instances) are available, there is a very good opportunity for matching systems. When two ontologies share the same set of individuals, matching is highly facilitated. For example, if two classes share exactly the same set of individuals, then there can be a strong presumption that these classes are equivalent.

Even when classes do not share the same set of individuals, these allow the grounding of the matching process on tangible indices which do not change easily. For instance, titles of Books do not have any reason to change. So if titles of Books are different, then these are most certainly not the same books. Then, matching can be again based on individual comparisons.

We thus divide extensional methods into three categories: those which apply to ontologies with common instance sets (Sect. 5.4.1), those which propose individual identification techniques, before using the previous ones (Sect. 5.4.2), and those which do not require identification, i.e., which work on heterogeneous sets of instances (Sect. 5.4.3).

5.4.1 Common Extension Comparison

The easiest way to compare classes when they share instances is to test the intersection of their instance set A and B and to consider that these classes are very similar when $A \cap B = A = B$, more general when $A \cap B = B$ or $A \cap B = A$. Relationships and entity sets can be integrated primarily based on the set relations: equal ($A \cap B = A = B$), contains ($A \cap B = A$), contained-in ($A \cap B = B$), disjoint ($A \cap B = \emptyset$) and overlap (Larson et al. 1989; Sheth et al. 1988). The problem is the ability to handle faults: small amounts of incorrect data may lead the system to draw a wrong conclusion on domain relationships. Moreover, the dissimilarity has to be 1 when none of these cases apply: for instance, if the classes have some instances in common but not all.

A way to refine this is to use the Hamming distance between two extensions: it corresponds to the size of the symmetric difference normalised by the size of the union.

Definition 5.36 (Hamming distance) The Hamming distance between two sets is a dissimilarity function $\delta : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$:

$$\delta(x, y) = \frac{|x \cup y - x \cap y|}{|x \cup y|}$$

This version of the symmetric difference is normalised. Using such a distance in comparing sets is more robust than using equality: it tolerates some individuals being misclassified and can still produce a short distance.

It is also possible to compute a similarity based on the probabilistic interpretation of the set of instances. This is the case of the Jaccard similarity (Jaccard 1901).

Definition 5.37 (Jaccard similarity) Given two sets A and B , let $P(X)$ be the probability of a random instance to be in the set X . The Jaccard similarity is defined as

$$\sigma(A, B) = \frac{P(A \cap B)}{P(A \cup B)}$$

This measure is normalised and reaches 0 when $A \cap B = \emptyset$ and 1 when $A = B$. It can be used with two classes of different ontologies sharing the same set of instances.

Formal Concept Analysis

One of the tools of formal concept analysis (FCA) (Ganter and Wille 1999) is the computation of the concept lattice. The idea behind formal concept analysis is the duality between a set of objects (here the individuals) and their properties: the more properties are constrained, the fewer objects satisfy the constraints. So, a set of objects with properties can be organised in a lattice of concepts covering these objects. Each concept can be identified by its properties (the intent) and covers the individual satisfying these properties (the extent).

In ontology matching, the properties can simply be the classes to which the individuals are known to belong and the technique is independent from the origin of the entities, i.e., whether they come from the same ontology or not. From this data set, formal concept analysis computes the concept lattice (or Galois lattice). This is performed by computing the closure of the instances \times properties Galois connection. This operation starts with the complete lattice of the power set of extent (respectively, intent) and keeps only the nodes which are closed under the connection, i.e., starting with a set of properties, it determines the corresponding set of individuals, which itself provides a corresponding set of properties; if this set is the initial one, then it is closed and is preserved, otherwise, the node is discarded. The result is a concept lattice.

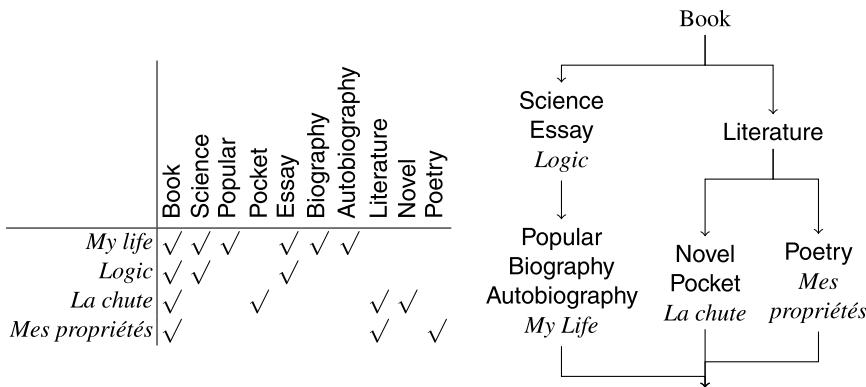


Fig. 5.4 A ‘formal context’ and the corresponding concept lattice.

The table of Fig. 5.4 displays a small set of instances and the classes they belong to (from both ontologies). The right-hand side of Fig. 5.4 displays the corresponding

concept lattice. From this lattice the following correspondences may be extracted:

$$\begin{array}{lll}
 \text{Science} = \text{Essay} & \text{Science} \geq \text{Biography} & \text{Essay} \geq \text{Popular} \\
 \text{Science} \geq \text{Autobiography} & \text{Popular} = \text{Biography} & \text{Popular} = \text{Autobiography} \\
 \text{Literature} \geq \text{Pocket} & \text{Novel} = \text{Pocket} &
 \end{array}$$

The result is not accurate. However, it is possible to weight these results by first eliminating the redundant correspondences and by providing a confidence according to the size of the extent covered by the correspondence.

5.4.2 Instance Identification Techniques

If a common set of instances does not exist, it is possible to try identifying which instance from one set corresponds to which other instance from the other set. This method is usable when one knows that the instances are the same. This works, for example, when integrating two human resource databases of the same company, but does not apply for those of different companies or for databases of events which have no relations.

These approaches have gained a lot of attention recently with the advent of linked data (Sect. 1.3). Indeed, data interlinking is an important task for linked data (Köpcke and Rahm 2010; Ferrara et al. 2011b), so many algorithms have been recently developed. It can benefit from ontology matching by using correspondences to focus the search for potential instance-level links (Nikolov et al. 2009; Scharffe and Euzenat 2011). In return, links produced by data interlinking provide an opportunity to identify the instances of ontologies to match. Thus matchers using links from the web of data were developed (Sect. 8.1.34). This is an illustration of the feedback loop presented in Fig. 1.5 (p. 12): better alignments provide better linked data, which in turn provide better alignments. Below, we briefly consider some such techniques.

Linkkey Extraction

A first natural technique for identifying instances is to take advantage of keys in data sets. Keys can be either internal to the data set, i.e., generated unique surrogates: in which case they are not very useful for identification, or external identification: in which case it is possible that these identification keys are present in both data sets (even if they are not present as keys). In such a case, if they are used as keys, we can be sure that they uniquely identify an individual (like isbn).

In general, what is really sought for are what we call *linkkeys*: sets of properties from both ontologies which, for a pair of classes, identify pairs of instances describing the same individual. Linkkeys should unambiguously identify entities from both

data sets. So, they should provide the correspondences between properties of both data sets to identify instances with the same values. They must be nonambiguous only on the instances selected by the correspondences, so there must be keys only for these sets.

Example 5.38 (Linkkey extraction) Consider two (small) book data sets described in the following tables:

bookstore data set				library data set					
	isbn	firstname	title	lastname	lang	year	author	orig	translator
3	E.		Gold bug	Poe	en	1845	Poe	Raven	Baudelaire
4	T.		On murder	Quincey	en	1845	Poe	Raven	Mallarmé
5	T.		Kant	Quincey	en	1843	Poe	Gold Bug	Baudelaire
6	T.		Confessions	Quincey	en	1827	Quincey	On murder	Schwob
7	J.-J.		Confessions	Rousseau	fr	1827	Quincey	Kant	Schwob
8	T.		Confessions	Aquinus	fr	1822	Quincey	Confessions	Musset

The keys for the bookstore data source are $\{\text{isbn}\}$, $\{\text{title}, \text{lastname}\}$, or $\{\text{title}, \text{firstname}, \text{lang}\}$ and, for the library data source, $\{\text{orig}, \text{translator}\}$.

A linkkey is $\langle\{\text{title}, \text{lastname}\}, \{\text{author}, \text{orig}\}, \{\text{title} = \text{orig}, \text{lastname} = \text{author}\}\rangle$. So, the linkkey uses a non-minimal key of the bookstore data set, because there is no counterpart to the minimal key $\{\text{isbn}\}$ in the library data set (and thus there is no correspondence involving isbn). More surprising, the linkkey does not use a key of the library data set, because (i) there is no counterpart to the translator property in the bookstore data set, and (ii) the set of properties used is sufficient for generating links nonambiguously (this would have been different if a tuple $\langle\text{year} = 1822, \text{author} = \text{Quincey}, \text{orig} = \text{Confessions}, \text{translator} = \text{Baudelaire}\rangle$ were present in the library data set).

It is possible to establish more elaborate linkkeys, such as $\langle\{\text{title}, \text{lastname}, \text{lang}\}, \{\text{title}, \text{author}\}, \{\text{lastname} = \text{author}, \text{title} = \text{translate}(\text{title}, \text{lang})\}\rangle$, which may be useful in case the orig column is not available. Such linkkeys may be useful when unit conversions are necessary.

As can be noted from the example, the third element of a linkkey is literally an alignment between properties. In principle, key extraction is performed by algebraic techniques isolating the smallest subsets of properties for which no two instances have the same values. Linkkeys can be treated in the same way, taking into account correspondences (equivalences) between the properties and replacing instances by pairs of instances. However, given the open and often nonperfect quality of linked data sets, it is useful to develop techniques for finding keys that maximise the support that they have in the data sets (Atencia et al. 2012b). Another difficulty when dealing with ontology extensions is the definition of keys when properties and relations may have several values.

Similarity-Based Instance Matching

When keys are not available, or they are different, other approaches to determine property correspondences use instance data to compare property values. In databases, this technique has been known as record linkage (Fellegi and Sunter 1969; Elfeky et al. 2002) or object identification (Lim et al. 1993). They aim at identifying multiple representations of the same object within a set of objects. They are usually based on string-based and internal structure-based techniques (Sects. 5.2 and 5.3).

Most of the data interlinkers dealing with instances nonidentifiable by keys work under the same schema (Ngomo and Auer 2011; Araújo et al. 2012): they first apply a phase of blocking (similar to the one described in Sect. 7.1.1) for identifying sets of instances that must be compared; then they use a similarity measure for comparing these instances and deciding if they represent the same entity or not. A framework like Silk (Sect. 12.4.2) allows for precisely describing these two steps. It offers a language for plugging and aggregating similarity measures. The considered similarity measures are typically those considered in this chapter. In addition, measures specific to the data types to consider have been developed, e.g., measures for matching geographic areas or addresses.

Since the data is usually described through ontologies, classes are a relevant first level of blocking. Hence, some data interlinkers, like KnoFuss (Sect. 12.4.1), can take as input an ontology alignment, which constrains classes whose instances will be compared.

Finally, given that the size of linked data is usually greater than that of ontologies, this makes learning the similarity more relevant for two reasons:

- The size of the data makes it difficult to study it for choosing the best approach, and after extracting a training sample, much work remains to be done;
- The regularity of the data facilitates machine learning efficiency.

So, it is not surprising that learning methods, such as genetic programming (Sect. 7.6.2), have been used in data interlinking (Ngomo 2011; Isele and Bizer 2013). If values are not precisely the same but their distributions can be compared, it is possible to apply global techniques. This case is covered in the next section.

5.4.3 Disjoint Extension Comparison

When it is not possible to directly infer a data set common to both ontologies, it is easier to use approximate techniques for comparing class extensions. These methods may be based on statistical measures about the features of class members, on the similarities computed between instances of classes or based on a matching between entity sets.

Statistical Approach

Instance data can be used to compute statistics about the property values found in instances, such as maximum, minimum, mean, variance, existence of null values, existence of decimals, scale, precision, grouping, and number of segments. This allows characterising the domains of class properties (Sect. 5.3.3) from the data. In practice, if dealing with statistically representative samples, these measures should be the same for two equivalent classes of different ontologies.

Example 5.39 (Statistical matching) Consider two ontologies with instances. The analysis of numerical properties size and weight in one ontology and hauteur and poids in the other reveals that they have different average values but the same coefficient of variation, i.e., standard deviation divided by mean, which, in turn, reveals comparable variability of size and hauteur on the one hand and weight and poids on the other hand. This is typically what happens when values are expressed in different units. The ratio of average values of size/hauteur is 2.54 and that of weight/poids is 28.35.

These values have been established based on the whole population. They can be used for comparing the statistical characteristics of these properties in the classes of the ontologies. For instance, the average value of the size property for the Pocket class significantly differs from that of the global population and, once divided by 28.35, is very close to that of the Livredepoche class (also differing from the whole population in the same manner). Hence, these two classes could be considered as similar.

Other approaches (Li and Clifton 1994) use data patterns and distributions instead of data values and domains. The result is a better fault tolerance and a lower time consumption since only a small portion of data values are needed due to the employment of data sampling techniques. In general, applying internal structure methods to instances allows for a more precise characterisation of the actual contents of schema elements, thus, more accurately determining corresponding data types based, for example, on the discovered value ranges and character patterns.

These methods have, however, one prerequisite: they work better if the correspondences between properties are known (otherwise they could match different properties on the basis of their domain). This is already a matching problem to be solved.

Similarity-Based Extension Comparison

Similarity-based techniques do not require classes to share the same set of instances, though they can still be applied in that case. In particular, the methods based on common extensions always return 0 when the two classes do not share any instance, disregarding the distance between the elements of the sets. In some cases, it is preferable to compare the sets of instances. This requires a (dis)similarity measure between the instances that can be obtained with the other basic methods.

In data analysis, linkage aggregation methods assess the distance between two sets whose objects are only similar. They thus allow for comparing two classes on the basis of their instances.

Definition 5.40 (Single linkage) Given a dissimilarity function $\delta : E \times E \rightarrow \mathbb{R}$, the single linkage measure between two sets is a dissimilarity function $\Delta : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$, $\Delta(x, y) = \min_{(e, e') \in x \times y} \delta(e, e')$.

Definition 5.41 (Complete linkage) Given a dissimilarity function $\delta : E \times E \rightarrow \mathbb{R}$, the complete linkage measure between two sets is a dissimilarity function $\Delta : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$, $\Delta(x, y) = \max_{(e, e') \in x \times y} \delta(e, e')$.

Definition 5.42 (Average linkage) Given a dissimilarity function $\delta : E \times E \rightarrow \mathbb{R}$, the average linkage measure between two sets is a dissimilarity function $\Delta : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$, $\Delta(x, y) = \frac{\sum_{(e, e') \in x \times y} \delta(e, e')}{|x| \times |y|}$.

Other linkage measures have been defined. Each of these methods has its own benefits, e.g., maximising shortest distance, minimising longest distance, minimising average distance. Another method from the same family is the Hausdorff distance measuring the maximal distance of a set to the nearest point in the other set (Hausdorff 1914):

Definition 5.43 (Hausdorff distance) Given a dissimilarity function $\delta : E \times E \rightarrow \mathbb{R}$, the Hausdorff distance between two sets is a dissimilarity function $\Delta : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$,

$$\Delta(x, y) = \max \left(\max_{e \in x} \min_{e' \in y} \delta(e, e'), \max_{e' \in y} \min_{e \in x} \delta(e, e') \right)$$

Matching-Based Comparison

The problem with the former distances, but average, is that their value is a function of the distance between one pair of members of the sets. The average linkage, on the other hand, has its value function of the distance between all the possible comparisons.

Matching-based comparisons (Valtchev 1999) consider that the elements to be compared are those which correspond to each other, i.e., the most similar one.

To that extent, the distance between two sets is considered as a value to be minimised and its computation is an optimisation problem: that of finding the elements of both sets which correspond to each others. In particular, it corresponds to solving a bipartite graph matching problem (Sect. 7.7.3).

Definition 5.44 (Match-based similarity) Given a similarity function $\sigma : E \times E \rightarrow \mathbb{R}$, the match-based similarity between two subsets of E is a similarity func-

tion $MSim : 2^E \times 2^E \rightarrow \mathbb{R}$ such that $\forall x, y \subseteq E$,

$$MSim(x, y) = \frac{\max_{p \in Pairings(x, y)} (\sum_{(n, n') \in p} \sigma(n, n'))}{\max(|x|, |y|)},$$

with $Pairings(x, y)$ being the set of mapping of elements of x to elements of y .

This match-based similarity already requires an alignment to be computed. It also depends on the kind of alignment that is required. Indeed, the result will be different depending on whether the alignment is required to be injective or not. The match-based comparison can also be used when comparing sequences (Valtchev 1999).

Summary on Extensional Techniques

Knowing extension information is invaluable for ontology matching because this provides information that is independent from the conceptual part of the ontology. Indeed, ontologies are views of the world and this is the reason why there can be numerous different ontologies on the same topic (and the reason why they have to be matched). Extension information is supposed to be less prone to variability and can be used to accurately match classes.

A very favourable case occurs when two ontologies to be matched share the same instances or when it is straightforward to connect the instances. This provides an easy way to compare the overlap between two classes. In most situations, however, the instance spaces are different. In such cases, it is either possible to match these instances in order to return to the previous case, or to use purely statistical techniques to compare classes on global measures over their extensions.

5.5 Summary

We have discussed basic techniques that can be used for building correspondences based on terminological (Sect. 5.2), internal structure (Sect. 5.3), and extensional (Sect. 5.4) arguments. This classification of techniques is a natural one since each of these deals with a partial view of ontologies.

There are many such techniques and our goal was not to present them all. It was rather to propose a panorama of the most used ones so far and to show the direction they take. There is still much work going on in finding better methods in each of these directions.

In general, these techniques are not used in isolation. They serve as the basis of more global methods or they are combined together for reenforcing their strengths. This is the topic of the two next chapters.

Chapter 6

Global Matching Methods

The basic similarities presented in Chap. 5 can be considered local because, in order to assess the similarity or dissimilarity between two entities, they only consider their proper characteristics (name, internal structure and extension). We consider here global methods, which consider the characteristics holding between the various entities in order to compare them.

We thus first consider the relational structure holding between the different entities (Sect. 6.1); this corresponds to methods using the external structure of entities as opposed to their internal structure (Sect. 5.3). When cycles occur in ontologies, as they often do, simple similarity computation is not possible. So, we present iterative similarity computation techniques (Sect. 6.2), which can deal with cycles and nonlinearity in the constraints governing relational similarities. These techniques tend to approximate an optimal solution for these constraints. Hence, we consider the use of classical optimisation techniques as a way to solve the matching problem (Sect. 6.3). Among global methods are probabilistic methods, which compute a probability of entities to be related based on the structure of ontologies and alignments (Sect. 6.4). Finally, we consider semantic techniques, because they depend on the global interpretation of each of the ontologies and alignments (Sect. 6.5).

These approaches often rely on basic matchers to provide seed alignments or anchors to be improved. For that reason, contrary to Chap. 1, in which a matcher was always a rectangle, each figure of this chapter and the next one details the inside of such a box, which is now represented as a dashed rectangle. Parameters and resources are omitted from the diagrams for the sake of readability. However, these exist and can be dispatched to all the subcomponents of the resulting matchers.

6.1 Relational Techniques

An ontology can be considered as a graph whose edges are labelled by relation names (mathematically speaking, this is the graph of the multiple relations of the ontology: \sqsubseteq , \in , \perp , $:$, $=$). Finding correspondences between elements of two such graphs corresponds to finding a common homomorphic subgraph of both graphs.

Definition 6.1 (Common homomorphic directed subgraph) Given two directed graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$, a *common homomorphic directed subgraph* of G and G' is a graph $\langle V'', E'' \rangle$, such that there exists a pair of homomorphisms $f : W \rightarrow V''$ and $g : W' \rightarrow V''$ with $W \subseteq V$ and $W' \subseteq V'$, such that

- $\forall w \in V'', \exists u \in V; f(u) = w$ and $\exists v \in V'; g(v) = w$;
- $\forall \langle u, v \rangle \in E|_{W \times W}, \langle f(u), f(v) \rangle \in E''$;
- $\forall \langle u', v' \rangle \in E'|_{W' \times W'}, \langle g(u'), g(v') \rangle \in E''$.

Graph matching is another type of problem, which is presented in Sect. 7.7.3.

The problem can be expressed as an optimisation problem. In graph theory, one would like to find a graph such that there is no other common homomorphic directed subgraph $\langle V''', E''' \rangle$ with $|V'''| > |V''|$ or with $V'' \subset V'''$. In ontology matching, these subgraphs do not have to be maximal: they will minimise some distance like the dissimilarity between matched objects or maximise similarity. Moreover, the problem is very often adapted for multipartite graphs, separating classes from properties.

The similarity comparison between two entities from two ontologies can be based on the relations of these entities with the other entities in the ontologies: the more two entities are similar, the more their related entities should be alike. This remark may be exploited in several ways depending on the kind of relations considered. Moreover, given the transitive nature of some relations, it is natural to extend this remark through transitivity. Roughly, for each pair of relations, we can come up with 5 different ways of comparing the relations (Euzenat et al. 2004):

- r comparing the entities in direct relation through r ;
- r^- comparing the entities in the transitive reduction of relation r ;
- r^+ comparing the entities in the transitive closure of relation r ;
- r^{-1} comparing the entities coming through a relation r ;
- r^\uparrow comparing entities which are ultimately in r^+ (the maximal elements of the closure).

These relations are illustrated as follows:

Example 6.2 (Exploiting relations in an ontology) Given the leftmost ontology of Fig. 2.7, the relations based on subClass from Book are as follows:

$$\begin{aligned} \text{subClass}(\text{Book}) &= \text{subClass}^-(\text{Book}) = \{\text{Science, Pocket, Children}\} \\ \text{subClass}^+(\text{Book}) &= \{\text{Science, Pocket, Textbook, Popular, Children}\} \\ \text{subClass}^{-1}(\text{Book}) &= \{\text{Product}\} \\ \text{subClass}^\uparrow(\text{Book}) &= \{\text{Textbook, Popular, Pocket, Children}\} \end{aligned}$$

Table 6.1 displays the different ways of comparing two ontology entities based on their relations with other entities. Of course, approaches may combine several of

Table 6.1 Features on which ontology entities can be compared. The table reads: Two *Entities* are similar if their *Features* are similar. This table is an adapted version of tables reported in (Ehrig 2007; Euzenat et al. 2004; Euzenat and Valtchev 2004).

Entity	Feature	OWL	Type
Class	name	rdf:label	String
	id	rdf:ID	String
	comments	rdf:comment	String
	same classes	owl:sameClassAs	Set(Class)
	properties	<i>property</i>	Set(Property)
	ultimate properties	<i>property</i> [↑]	Set(Property)
	direct superclasses	owl:subClassOf ⁻	Set(Class)
	direct subclasses	owl:subClassOf ⁻¹⁻	Set(Class)
	superclasses	owl:subClassOf [*]	Set(Class)
	subclasses	owl:subClassOf ^{-1*}	Set(Class)
	ultimate subclasses	owl:subClassOf ⁻¹ [↑]	Set(Class)
	direct instances	rdf:type ^{-1*}	Set(Individual)
	instances	rdf:type ⁻¹⁻	Set(Individual)
Property	name	rdf:label	String
	id	rdf:ID	String
	comments	rdf:comment	String
	same properties	owl:samePropertyAs	Set(Property)
	domain/range	rdfs:domain/rdfs:range	Class
	direct superproperties	rdfs:subProperty ⁻	Set(Property)
	direct subproperties	rdfs:subProperty ⁻¹⁻	Set(Property)
	superproperties	rdfs:subProperty [*]	Set(Property)
	subproperties	rdfs:subProperty ^{-1*}	Set(Property)
Individual	name	rdf:label	String
	id	rdf:ID	String
	comments	rdf:comment	String
	same individuals	owl:sameAs	Set(Instance)
	direct classes	rdf:type ⁻	Set(Class)
	classes	rdf:type [*]	Set(Class)
	properties	<i>property</i>	Set(Property)

the above criteria (Mädche and Staab 2002; Euzenat and Valtchev 2004; Bach et al. 2004).

As can be observed from Table 6.1, some features have type String and can be compared with the techniques proposed in Sect. 5.2.1. However, those with type Class or Property really induce a graph structure. Moreover, the values which are labelled by Set(\cdot) are more difficult to deal with because this means that many edges

labelled by the feature will appear in the graph. The last part of the table is, in fact, relevant to the extensional methods presented in Sect. 5.4.

There are three types of relations that have been considered so far in relational structure techniques: taxonomic relations, mereologic relations and all the involved relations. These are considered below.

6.1.1 Taxonomic Structure

The taxonomic structure, i.e., the graph made with the `subClassOf` relation, is the backbone of ontologies. For this reason, it has been studied in detail by researchers and is very often used as a comparison source for matching classes.

There have been several measures proposed for comparing classes based on the taxonomic structure. The most common ones are based on counting the number of edges in the taxonomy between two classes. The structural topological dissimilarity on a hierarchy (Valtchev and Euzenat 1997) follows the graph distance, i.e., the shortest path distance in a graph taken here as the transitive reduction of the hierarchy.

Definition 6.3 (Structural topological dissimilarity on hierarchies) The structural topological dissimilarity $\delta : o \times o \rightarrow \mathbb{R}$ is a dissimilarity over a hierarchy $H = \langle o, \leq \rangle$, such that

$$\forall e, e' \in o, \quad \delta(e, e') = \min_{c \in o} [\delta(e, c) + \delta(e', c)]$$

where $\delta(e, c)$ is the number of intermediate edges between an element e and another element c .

This corresponds to the unit tree distance, i.e., with weight 1 on each edge (Barthélemy and Guénoche 1992). This function can be normalised by the maximal length of a path between two classes in the taxonomy:

$$\bar{\delta}(e, e') = \frac{\delta(e, e')}{\max_{c, c' \in o} \delta(c, c')}$$

Example 6.4 (Structural topological dissimilarity) We provide the examples of this section based on the taxonomy in Fig. 5.1. We consider that each term corresponds to a class (all senses are considered together) and there exists a top of the hierarchy (on top of Person, literate, legal document and God).

	illustrator	author	creator	Person	writer
illustrator	0.	.8	.4	.6	1.
author	.8	0.	.4	.6	0.
creator	.4	.4	0.	.2	.6
Person	.6	.6	.2	0.	.4
writer	1.	0.	.6	.4	0.

Again, this corroborates the WordNet data that the closest classes are writer and author.

The results given by such a measure are not always semantically relevant since a long path in a class hierarchy can often be summarised as an alternative short one.

A similar measure is the one of Leacock–Chodorow (Leacock et al. 1998), which is function of the length of the shortest path. It was introduced for lexicographic taxonomies (Sect. 5.2.2). A more elaborate measure of this kind is known as the Wu–Palmer similarity (Wu and Palmer 1994). This similarity measure takes into account the fact that two classes near the root of a hierarchy are close to each other in terms of edges but can be very different conceptually, while two classes under one of them, which are separated by a larger number of edges, should be closer conceptually. The initial Wu–Palmer similarity was designed by counting nodes as the length of a path, instead of using the usual graph-theoretic edge counting. We prefer the formulation in terms of edges (Resnik 1999). When used in ontology matching, the difference is not relevant.

Definition 6.5 (Wu–Palmer similarity) The Wu–Palmer similarity $\sigma : o \times o \rightarrow \mathbb{R}$ is a similarity over a hierarchy $H = \langle o, \leq \rangle$, such that

$$\sigma(c, c') = \frac{2 \times \delta(c \wedge c', \rho)}{\delta(c, c \wedge c') + \delta(c', c \wedge c') + 2 \times \delta(c \wedge c', \rho)}$$

where ρ is the root of the hierarchy, $\delta(c, c')$ is the number of edges between a class c and another class c' included and $c \wedge c' = \{c'' \in o; c \leq c'' \wedge c' \leq c''\}$.

Example 6.6 (Wu–Palmer similarity) The Wu–Palmer similarity also provides a figure in coherence with the WordNet structure.

	illustrator	author	creator	Person	writer
illustrator	1.	.5	.67	.4	.29
author	.5	1.	.67	.4	1.
creator	.67	.67	1.	.67	.4
Person	.4	.4	.67	1.	.5
writer	.29	1.	.4	.5	1.

The upward cotopic similarity applies the Jaccard similarity to cotopies. It has been described in (Mädche and Zacharias 2002) and is defined as follows:

Definition 6.7 (Upward cotopic similarity) The upward cotopic similarity $\sigma : o \times o \rightarrow \mathbb{R}$ is a similarity over a hierarchy $H = \langle o, \leq \rangle$, such that

$$\sigma(c, c') = \frac{|UC(c, H) \cap UC(c', H)|}{|UC(c, H) \cup UC(c', H)|}$$

where $UC(c, H) = \{c' \in H; c \leq c'\}$ is the set of superclasses of c ($superclass^+(c)$).

Example 6.8 (Upward cotopic similarity) In this case, because all senses count in the cotopy (and not the closest one in terms of path), the result is different from other measures: creator benefits from its position as a superclass of author and illustrator for scoring better than the usual writer-creator pair because they have too many unrelated senses.

	illustrator	author	creator	Person	writer
illustrator	1.	.37	.43	.4	.18
author	.37	1.	.43	.29	.36
creator	.43	.43	1.	.4	.18
Person	.4	.29	.4	1.	.25
writer	.18	.36	.18	.25	1.

These measures cannot be applied as they are in the context of ontology matching since the ontologies are not supposed to share the same taxonomy H , but this can be used in conjunction with a resource of common knowledge, such as WordNet. For that purpose, it is necessary to develop these kinds of measures over a pair of ontologies. In (Valtchev 1999; Euzenat and Valtchev 2004), this uses a (local) matching between the elements to be compared (for instance, the hierarchies).

Besides these global measures that take into account the whole taxonomy for assessing the similarity between classes, nonglobal measures have been used in ontology matching. These measures usually take advantage of the ‘direct’ part of Table 6.1. Below are some of these measures:

Super or subclass rules: Some matchers are based on rules capturing the intuition that classes are similar if their super or subclasses are similar. For example, if superclasses are the same, the actual classes are similar to each other. If subclasses are the same, the compared classes are also similar (Dieng and Hug 1998; Ehrig and Sure 2004). This technique has at least two drawbacks: (i) when there are several sub or superclasses, then, without care, they would all be mapped into the same one, so it is necessary to have some other discriminating features, and (ii) the similarity between the sub or superclasses will rely in turn on that of their super or subclasses. This turns this problem into yet another global similarity problem.

Bounded path matching: Bounded path matchers take two paths with links between classes defined by the hierarchical relations, compare terms and their positions along these paths, and identify similar terms. This technique has been

introduced in Anchor-Prompt (Sect. 8.1.9). For example, in Fig. 2.9, if Book corresponds to Volume and Popular corresponds to Autobiography, then the elements along the paths (Science on the one hand and Biography and Essay on the other hand) must be carefully considered for correspondence. For instance, for deciding that Essay is more general than Science. This technique is primarily guided by two anchors of paths and uses alternative techniques for choosing the best match.

In addition, edit distances (Sect. 5.2.1) can also be applied to other structures than strings, and in particular to tree (Tai 1979).

6.1.2 Mereologic Structure

The second well-known structure after the taxonomic structure is the mereologic structure, i.e., the structure corresponding to a *part-of* relationship. If two classes Book and Volume have been found equivalent, and both have mereologic relations with classes InBook and BookChapter, respectively, then this is a suggestion that these two classes may be related as well. This inference also applies in the reverse direction: from the part to the whole. This may be generalised when there are individualised parts, i.e., when the parts of a journal Issue are differentiated into editorials, articles, recensions and letters.

The difficulty for dealing with this kind of structure is that it is not easy to find the properties which carry a mereologic structure. For example, a class Proceedings can have some whole-part relations with a class InProceedings, but it will be expressed through a property communications. These InProceedings objects will in turn have a mereologic structure which is expressed through sections property.

However, if it is possible to detect the relations that support the mereologic structure, this can be then used for computing similarity between classes: they will be more similar if they share similar parts. This is even more useful when comparing extensions of classes because it can be inferred that objects sharing the same set of parts will be the same.

6.1.3 Relations

Besides these two kinds of relations, one may consider the general problem of matching entities based on all their relations. Classes are also related through the definitions of their properties (like author and creator in Fig. 5.2). These properties are edges of a graph and if they are found similar, they can be used for finding that classes are similar. However, contrary to taxonomic and mereologic structures, the relation graph may contain circuits. How to handle these will be considered in Sect. 6.2. We consider here similarities.

The similarity between nodes can also be based on their relations. For example, in one of the possible ontology representations of schemas of Fig. 2.7, if the Book class is related to the Human class by the author relation in one ontology, and if the Volume class is related to the Writer class by the author relation in the other ontology, then knowing that classes Book and Volumes are similar, and that relations author and author are similar, we can infer that Human and Writer may be similar too. The similarity among relations in (M  dche and Staab 2002) is computed according to this principle.

This can be applied to a set of classes and a set of relations. If we have a set of relations $r_1 \dots r_n$ in the first ontology which are similar to another set of relations $r'_1 \dots r'_n$ in the second ontology, it is possible that two classes, that are the domains of relations in those two sets, are similar too. Discrimination between possible such correspondences may be obtained by retaining those supported by the largest number of properties.

This principle can also be extended to the composition of relations, i.e., instead of considering only the relations asserted at a class, one may consider their composition with relations starting at the domain of this relation. For instance, instead of considering the author relation, one will consider the author·firstname, the author·lastname, or the author·nationality relations.

One of the problems of this approach is that it is based on the use of similarity of relations to infer the similarity of their domain classes or their range classes. This introduces circularity in the computation of similarity. There are several ways to overcome this circularity. As a first alternative, the similarity on relations can be based on their labels using techniques developed in Sect. 5.2.1. As a second alternative, if relations are organised in a taxonomy, then methods considered in the previous subsection may be used as well.

Finally, two extreme solutions, that use the relations for reaching nodes but not for actually matching, are considered by the following approaches:

Children. The similarity between nodes of the graph is computed based on similarity of their children nodes, that is, two nonleaf entities are structurally similar if their immediate child sets are highly similar. A more complex version of this matcher is described in (Do and Rahm 2002).

Leaves. The similarity between nodes of the graphs is computed based on similarity of leaf nodes, that is, two nonleaf entities are structurally similar if their leaf sets are highly similar, even if their immediate children are not (Madhavan et al. 2001; Do and Rahm 2002). This is very well adapted to comparing document schemas.

6.1.4 Pattern-Based Matching

Pattern-based matching consists of finding correspondences by identifying corresponding patterns in two ontologies. This starts from a set of predefined correspon-

dence patterns (see Fig. 6.1). A correspondence pattern is an abstraction of a correspondence, i.e., it does not relate individual entities, but instead types of entities. Such patterns typically identify alternative concept modelling patterns.

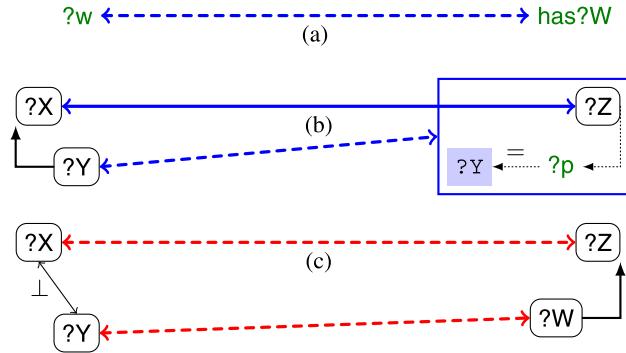


Fig. 6.1 Three correspondences patterns: (a) property $?w$ is equivalent to property $\text{has}?W$, (b) a subclass $?Y$ may correspond to the elements whose value of property $?p$ is $?Y$, (c) an antipattern such that if $?X$ is disjoint from $?Y$ and $?W$ is a subclass of $?Z$, it is forbidden that both $?X$ is equivalent to $?Z$ and $?Y$ is equivalent to $?W$.

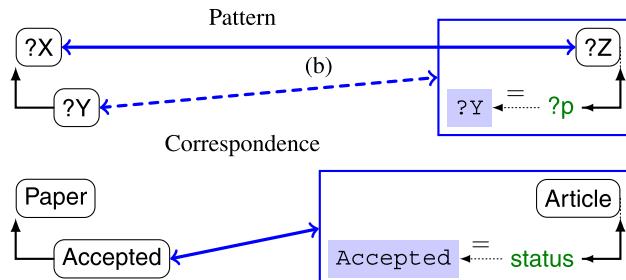


Fig. 6.2 Pattern instantiation: an instance of a correspondence pattern is a correspondence (here in EDOAL).

(Scharffe and Fensel 2008) introduced correspondence patterns and their use in ontology matching as instantiating them in ontologies. For instance, in Fig. 6.2, the assignment $\{?X \rightarrow \text{Paper}, ?Y \rightarrow \text{Accepted}, ?Z \rightarrow \text{Article}, ?p \rightarrow \text{status}\}$ allows for identifying the patterns within ontologies and for considering the result as a candidate correspondence. (Scharffe 2008) proposed a library of 44 correspondence patterns based on a study of ontology mismatches. In order to be meaningful, correspondence patterns have to use an expressive language, either on the side of ontologies or on the side of alignment language. The EDOAL language (Sect. 10.1.6) was defined partly with this goal in mind.

Alignment patterns, called block matching patterns (Šváb-Zamazal 2010), involve, on the same principle, several correspondences together. It also introduced more tolerance in patterns through the use of similarity measures, usually based on string similarities for deciding the instances of patterns. Patterns may also involve constraints on values, such as $v = u \pm 3\%$. In turn, (Ritze et al. 2009) experimented with specific algorithms for recognising four specific such patterns.

Antipatterns (Roussey et al. 2009) also identify correspondences that should *not* be included in alignments (see Fig. 6.1). The rules used by systems for detecting inconsistency or incoherence in alignments may be (semantic) alignment antipatterns (Sect. 7.8.2).

Refinement pattern, expressed in a first-order language, are made of an antipattern and a pattern (Hamdi et al. 2010a). The method detects, in the same alignment, the presence of the antipattern and the possibility of instantiating the pattern. This is then used to propose users to replace the former by the latter. Other work defines patterns not simply with respect to the ontology language, i.e., identifying classes or properties, but with respect to a common foundational ontology, i.e., identifying events or participants (Padilha et al. 2012). Pattern-based matching is classified as a structural technique because it matches the structure. However, this particular structure can also be detected through reasoning.

Summary on Relational Techniques

Matching ontologies from their relational (or external) structure is very powerful because it allows all the relations between entities to be taken into account. This must be grounded on other tangible properties, which is why it is often used in combination with internal structural methods and terminological methods.

It is worth considering what are the important relations before using such techniques. The most commonly used structure is the specialisation taxonomy because it is the backbone of ontologies and has usually received a lot of attention from designers. In some fields, mereologic relations are as important as taxonomic ones. However, they are difficult to identify because contrary to the `subClass` relation, they can bear any other name.

The relational structure raises the problem of which part influences what: there is usually a mutual influence between each of the related parts. This is the reason why, besides the similarity equations used for comparing the entities, it is necessary to have an iterative algorithm. This is considered in the next section.

6.2 Iterative Similarity Computation

The computation of compound similarity is still local because it only provides similarities by considering the neighbourhood of a node. However, similarity may involve the ontologies as a whole and the final similarity values may ultimately depend on all the ontology entities. Moreover, the distance defined by local methods

can be defined in a circular way when the ontology is not reduced to a directed acyclic graph. This is the most common case. For instance, this occurs if the distance between two classes depends on the distances between their instances which themselves depend on the distance between their classes or if there are circuits in the ontology. This is illustrated in Fig. 6.3, in which the similarity between Product and Book depends on the similarity between hasProvider and hasCreator and author, publisher, and translator. In turn, the similarity between these elements ultimately depends on the similarity between Product and Book. These two graphs are homomorphic in many different ways.

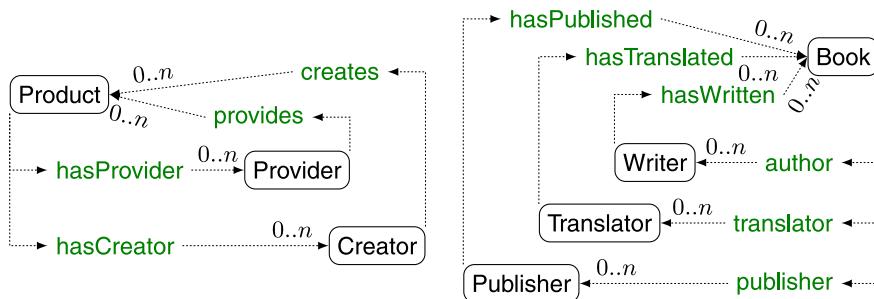


Fig. 6.3 Two typical ontologies containing referential cycles: how do we match them?

In case of circular dependencies, similarity computation in a local fashion is no longer possible. The classical way of dealing with such a problem involves the iterative computation of the distance or similarity refining at each step the last computed values. This is depicted in Fig. 6.4.

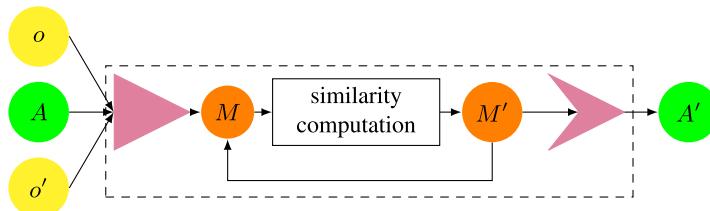


Fig. 6.4 Iterative computation of the fixed point of a similarity or distance function.

For that purpose, strategies must be defined in order to compute this global similarity. We present two such methods here. The first one is defined as a process of propagating the similarity within a graph (Sect. 6.2.1) while the second one translates the similarity definitions in a set of equations which is solved by numerical analysis techniques (Sect. 6.2.2).

6.2.1 Similarity Flooding

Similarity flooding (Melnik et al. 2002) is a generic graph matching algorithm which uses fixed point computation to determine corresponding nodes in the graphs. It is implemented in the Rondo environment (Sect. 10.3.1).

The principle of the algorithm is that the similarity between two nodes must depend on the similarity between their adjacent nodes (whatever are the relations that must be taken into account). To implement this, the algorithm proceeds as follows:

1. Transform the ontologies in a directed labelled (multi)graph G in which nodes are pairs of concepts of the ontologies and edges exist between two nodes if there is a relation in both ontologies between the nodes of the two pairs. For instance, in the ontology of Fig. 6.3, $\langle \text{Provider}, \text{Writer} \rangle$ is related to $\langle \text{Product}, \text{Book} \rangle$ through an edge labelled $\langle \text{hasProvider}, \text{hasWritten} \rangle$. In fact, the original Similarity flooding algorithm only connects nodes whose edges have the same label. The graph is closed by symmetry, i.e., there will also be an edge in the reverse direction.
2. Assign weights w to the edges, which are usually $1/n$ in which n is the out degree (the number of outgoing edges) of the source node. The algorithm description does not tell what to do when several edges with different labels link the same pair of concepts or when there is already a reverse edge. One can imagine that the weights are aggregated with a triangular norm (Sect. 7.4).
3. Assign initial similarity σ^0 to each node (with some basic method on labels of Sect. 5.2 or with a uniform assignment of 1.0).
4. Compute σ^{i+1} for each node with the chosen formula.
5. Normalise all σ^{i+1} obtained by dividing by the largest value.
6. If no similarity changes more than a particular threshold ϵ , i.e., $\forall e \in o, e' \in o', |\sigma^{i+1}(e, e') - \sigma^i(e, e')| < \epsilon$, or after a predetermined number of steps, stop; otherwise, go to step 4.

The chosen aggregation function is a weighted linear aggregation in which the weight of an edge is the inverse of the number of other edges with the same label reaching the same pair of entities.

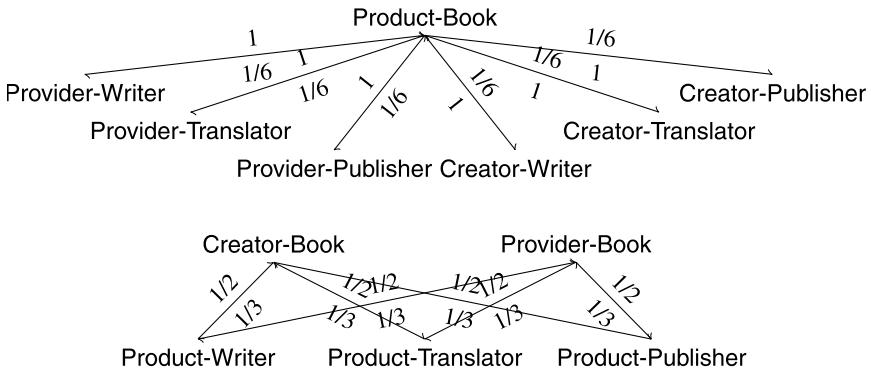
$$\sigma^{i+1}(x, x') = \sigma^0(x, x') + \sum_{\langle \langle y, y' \rangle, p, \langle x, x' \rangle \rangle \in G} \sigma^i(y, y') \times w(\langle \langle y, y' \rangle, p, \langle x, x' \rangle \rangle)$$

Several variations of this formula have been studied, including suppressing the σ^0 term and replacing σ^0 by σ^i , or using $\sigma^0(x, x') + \sigma^i(x, x')$ as the recurrence term. The former accelerates computation, while the latter gives more importance to the initial values.

The convergence of the algorithm is not obvious. (Melnik et al. 2005) provides conditions under which the algorithm converges. This algorithm does provide a similarity measure from which an alignment remains to be extracted (Sect. 7.7).

Example 6.9 (Similarity flooding) We start with the ontologies of Fig. 6.3. Since the Similarity flooding algorithm works with the same property names and there is

no similar property, we choose to consider that all properties have the same name. From these ontologies is generated the following labelled directed graph (with its weights):



The initial dissimilarity is the one provided in Example 7.9 for the weighted sum with weights of $1/4$ and $3/4$ respectively.

σ^0	Book	Translator	Publisher	Writer
Product	.19	.05	.03	.04
Provider	.04	.05	.75	.14
Creator	.04	.25	.04	.69

The first iteration of the Similarity flooding algorithm computing σ^1 is below (on the left are the σ^i values and on the right is the normalised result):

σ^1	Book	Translator	Publisher	Writer	$\bar{\sigma}^1$	Book	Translator	Publisher	Writer
Product	2.11	0.08	0.06	0.07	Product	1.00	0.04	0.03	0.03
Provider	0.10	0.08	0.78	0.17	Provider	0.05	0.04	0.37	0.08
Creator	0.10	0.28	0.07	0.72	Creator	0.05	0.13	0.03	0.34
σ^1					Normalised σ^1				

The iterative procedure carries on and, with a value of $\epsilon = .1$, stops at the 17th iteration with the following result:

σ^{17}	Book	Translator	Publisher	Writer	$\bar{\sigma}^{17}$	Book	Translator	Publisher	Writer
Product	1.95	.09	.07	.08	Product	1.00	.05	.04	.04
Provider	.11	.22	.92	.31	Provider	.06	.11	.47	.16
Creator	.11	.42	.21	.86	Creator	.06	.21	.11	.44
σ^{17}					Normalised σ^{17}				

From these similarity values, it is possible to extract the expected correspondences: Product = Book, Publisher = Provider and Writer = Creator.

6.2.2 Similarity Equation Fixed Point

OLA (Sect. 8.3.8) (Euzenat and Valtchev 2004) provides a method for dealing with circularities and dependencies between similarity definitions.

In this case, the similarity values are expressed as a set of equations in which each variable corresponds to the similarity between a pair of nodes. There are as many equations as variables. The structure of each equation follows the definition of the respective similarity function for the underlying node category.

Given two classes c and c' , the resulting class similarity function is

$$\begin{aligned}\sigma_C(c, c') = & \pi_L^C \sigma_L(\lambda(c), \lambda(c')) \\ & + \pi_O^C MSim_O(\mathcal{I}(c), \mathcal{I}'(c')) \\ & + \pi_S^C MSim_C(\mathcal{S}(c), \mathcal{S}'(c')) \\ & + \pi_P^C MSim_P(\mathcal{A}(c), \mathcal{A}'(c'))\end{aligned}$$

such that $\lambda(\cdot)$, $\mathcal{I}(\cdot)$, $\mathcal{S}(\cdot)$, $\mathcal{A}(\cdot)$ are the functions returning respectively the label, instances, super and subclasses, and properties of a class (corresponding to name, properties, direct subclasses, direct superclasses and direct instances in Sect. 6.1). $MSim$ -measures are similarities between sets of ontology entities, which we explain below. The $\pi_{\mathcal{F}}^C$ are weights denoting the relative importance of the feature \mathcal{F} .

The function is normalised since the sum of weights is equal to one, i.e., $\pi_L^C + \pi_S^C + \pi_O^C + \pi_P^C = 1$, whereas each factor that ranges over collections of nodes or feature values is averaged by the size of the largest collection.

If each of the similarity expressions were a linear aggregation of other similarity variables, this system would be solvable directly, since all variables are of degree one. However, in the case of OWL, and of many other languages, the system is not linear since there could be many candidate pairs for the best match. These correspond to the $Set(\cdot)$ type in Table 6.1 (p. 123). The similarity may depend on matching the multiple edges with the similar labels outgoing from the nodes under consideration. In this approach, the similarity is computed by a $MSim$ function that first finds an alignment between the set of considered entities and then computes the aggregated similarity with respect to this matching.

In this respect, the OLA algorithm solves a very specific problem, namely a maximal weight graph matching problem (Sect. 7.7.3) with weights depending on the matching.

Nevertheless, the resolution of the resulting system can still be carried out as an iterative process that simulates the computation of the greatest fixed point of a vector

function, as shown by (Bisson 1992). The point consists of defining an approximation of the *MSim*-measures, solving the system, replacing the approximations by the newly computed solutions and iterating. The first values for these *MSim*-measures are the maximum similarity found for a pair, without considering the dependent part of the equations. The subsequent values are those of the complete similarity formula filled by the solutions of the system. The local matching may change from one step to another depending of the similarity values.

However, the system converges because similarities can only increase (the independent part of the equation remains and all dependencies are positive) and, if similarity values are bounded, e.g., by 1, the similarity is bounded. The iterations will stop when no gain above a particular ϵ value is provided by the last iteration. If the algorithm converges, it may stop at a local optimum (that is, finding another matching in the *MSim*-measures would not increase the similarity values). This could be improved by randomly changing these matchings when the algorithm stops.

Some facts are worth mentioning. First, there is no need for a different expression of the similarity functions in the case where there are no effective circular dependencies between similarity values. The computation mechanism presented above establishes the correct similarity values even if there is an appropriate ordering of the variables (the ordering is implicitly followed by the stepwise mechanism). Moreover, in case some similarity values (or some similarity or (dis)similarity assertions) are available beforehand, the corresponding equation can be replaced by the assertion or value.

Example 6.10 (OLA algorithm) The problem to be solved is the same as the one defined in Example 6.9, so the label similarity between classes is the same. The label similarity between properties is set to 1. (all similar) for each pair of properties. Thus, the initial similarities are as follows:

σ_L	Book	Translator	Publisher	Writer
Product	.19	.05	.03	.04
Provider	.04	.05	.75	.14
Creator	.04	.25	.04	.69

σ_L	hasPublished	hasTranslated	hasWritten	author	translator	publisher
creates	1.00	1.00	1.00	1.00	1.00	1.00
provides	1.00	1.00	1.00	1.00	1.00	1.00
hasProvided	1.00	1.00	1.00	1.00	1.00	1.00
hasCreated	1.00	1.00	1.00	1.00	1.00	1.00

The equations are made with equal weights on labels and properties for classes ($\pi_L^C = \pi_P^C = 1/2$) and equal weights on label, range and domain for properties ($\pi_L^P = \pi_R^P = \pi_D^P = 1/3$). The initial similarities (based only on the labels) provide the following values:

σ^0	Book	Translator	Publisher	Writer
Product	.10	.03	.02	.02
Provider	.02	.03	.38	.07
Creator	.02	.13	.02	.35

σ^0	hasPublished	hasTranslated	hasWritten	author	translator	publisher
creates	.33	.33	.33	.33	.33	.33
provides	.33	.33	.33	.33	.33	.33
hasProvided	.33	.33	.33	.33	.33	.33
hasCreated	.33	.33	.33	.33	.33	.33

The first iteration really takes into account the relations between entities and yields the following result:

σ^1	Book	Translator	Publisher	Writer
Product	.26	.19	.18	.19
Provider	.19	.19	.54	.24
Creator	.19	.29	.19	.51

σ^1	hasPublished	hasTranslated	hasWritten	author	translator	publisher
creates	.37	.41	.48	.35	.35	.35
provides	.49	.37	.39	.35	.35	.35
hasProvided	.35	.35	.35	.39	.37	.49
hasCreated	.35	.35	.35	.48	.41	.37

After 3 iterations the values do not change more than $\epsilon = .1$ and after 10 iterations they do not change more than $\epsilon = .01$ yielding the result as follows:

σ^{10}	Book	Translator	Publisher	Writer
Product	.46	.29	.27	.28
Provider	.28	.32	.74	.37
Creator	.28	.44	.31	.70

σ^{10}	hasPublished	hasTranslated	hasWritten	author	translator	publisher
creates	.59	.63	.72	.52	.52	.52
provides	.73	.59	.61	.52	.52	.52
hasProvided	.52	.52	.52	.61	.59	.73
hasCreated	.52	.52	.52	.72	.63	.59

For both values of ϵ the best match is always the same. It is the same as in Example 6.9 for classes and, in addition, for properties it is as follows: creates = hasWritten, provides = hasPublished, hasProvided = publisher and hasCreated = author.

The two presented methods share some similarity: both methods work iteratively on a set of equations extracted from a graphical form of the ontologies. Both methods ultimately depend on the computed proximities between nondescribed language elements, i.e., data type names, values, URIs, property type names, etc. These proximities are propagated throughout the graph structure by the similarity dependencies.

Moreover, Similarity flooding is highly dependent on the identity of edge labels, while the OLA algorithm takes similarity between properties into account. Nonetheless, it also considers local mappings between alternative matching edges instead of averaging over all the potential matches. That is, the OLA algorithm attempts to identify the subclasses which match and propagate their similarity—which should be high—while Similarity flooding propagates an average similarity between all pairs of subclasses which should be lower than the average similarity between all pairs of matching subclasses. Finally, the convergence of Similarity flooding is not proved in general.

Summary on Global Similarity Computation

The intricacy of ontologies mandates propagating similarity values across them in an iterative fashion. There may be different propagation modalities depending on what the compound similarity definition is and along which connections links are propagated. As can be observed from the last technique, global similarity computation can be seen as an optimisation problem. It is thus natural to consider classical optimisation algorithms for achieving them.

6.3 Matching as Optimisation

In the previous section we presented iterative similarity computation by using sets of equations expressing similarities. This process can be considered as a simple iterative process (Sect. 6.2.1) or as an optimisation problem (Sect. 6.2.2). In the last case, classical optimisation techniques can be used for ontology matching. We consider here two such techniques: expectation maximisation (Sect. 6.3.1) and particle swarm optimisation (Sect. 6.3.2).

6.3.1 *Expectation Maximisation*

Expectation maximisation (EM) is an iterative approach to the maximum likelihood estimation problem (Dempster et al. 1977). It is typically used for parameter estimation (of a parametric probability distribution) when observable data is missing or partial, such as in hidden Markov models, to perform maximum a posteriori inference (Sect. 6.4.2). The missing data is estimated given the observed data through

the *conditional expectation* probability. In other words, the missing data is enhanced with guesses of potentially useful information over it. Then, the likelihood function is maximised assuming that the missing data is known. Thus essentially, each EM iteration consists of two steps: *expectation*, called the *E-step*, and *maximisation*, called the *M-step*, which maximises consecutive local improvements of a likelihood function approximated within the E-step. This is a fixed point method and convergence is guaranteed by increasing the likelihood at each iteration.

Example 6.11 (Expectation maximisation) The ontology matching problem was treated as maximum likelihood estimation in (Doshi et al. 2009; Thayasilvam and Doshi 2011). The method searches for the maximum likelihood estimate of the alignment from observed data in the presence of missing correspondences, which are treated as hidden variables. Let M be a binary match matrix, where rows and columns stand for the entities of the ontologies to be matched and the corresponding cells are populated with '1' standing for the match and '0', otherwise. The method iteratively searches for the match matrix M that provides the maximum conditional probability $P(o|o', M)$ of the source ontology o , given the target one o' and the match assignment made through the match matrix M , e.g., with initial correspondences. In EM terms, each match assignment variable is viewed as a model, and thus, the match matrix is viewed as a mixture model. Assuming that X is a set of observed data instances, Y is a set of missing values and M is the match matrix or the mixture model. The two EM steps can be formalised as follows:

E-step During the E-step, a weighted sum of the log-likelihood is computed:

$$Q(M^{i+1}|M^i) = \sum_{y \in Y} P(y|X, M^i) L(M^{i+1}|X, y),$$

such that i is the iteration number, the weights are the conditional probabilities of the hidden variables (correspondences) and $L(M^{i+1}|X, y)$ is the log-likelihood of the model in the iteration $i + 1$. The logarithm is used to simplify the computation, since the maximum of both L and $\log(L)$ is the same. Initial match seeds and probabilities of hidden variables can be estimated through terminological similarities or simple structural heuristics, such as ‘if two entities match, their parent nodes are likely to match as well’.

M-step During this step, the parameters that maximise the expected log-likelihood (of the E-step) are computed and the match matrix (mixture model) that maximises the expectation is selected. In practice, the maximisation condition is often relaxed by selecting a mixture model that just improves over the previous one, what is known as the *generalised* expectation maximisation method:

$$M_*^{i+1} \in \{M : Q(M|M^i) \geq Q(M^i|M^i)\}$$

The match matrix is then taken as input to determine the distribution of the hidden variables in the next E-step. Thus, the EM method revises iteratively the match

matrices (mixture models) by maximising a weighted sum of the log-likelihood of these models.

6.3.2 Particle Swarm Optimisation

Particle swarm optimisation (PSO) is a non-deterministic optimisation method belonging to the swarm intelligence family of methods (Engelbrecht 2005). This is a metaheuristic with *anytime* behaviour, i.e., when interrupted at any time, it provides the best result found so far. It was inspired by the collective intelligence of groups of simple agents, called particles, such as bird flocks or fish schools, adapting to the environments they live in (Kennedy and Eberhart 1995).

By analogy, given an initial swarm of randomly generated solutions, particles memorise and share the best solution they have seen. A particle swarm moves (iterates) over the solution space based on this shared information on the space; particles are evaluated based on a fitness criterion. Thus, the swarm explores the solution space by looking for promising areas. In particular, movements of particles are formalised through *velocity* vectors. These vectors (in a simplified view) are based on the individual best position (*pBest*) of particles, also known as the *personal influence* component, and/or on the result of the whole group of particles (*gBest*), also known as the *social influence* component. At each iteration, particles change velocities towards a better position based on *pBest* and/or *gBest*. Thus, the evolution searches only for the best solution. Unlike genetic algorithms (Sect. 7.6.2), PSO does not have genetic operations, such as crossover and mutation. Instead, particles are updated through velocity vectors.

Example 6.12 (Particle swarm optimisation) Ontology matching may be viewed as an optimisation problem that is attacked through a *discrete* particle swarm optimisation (DPSO) solution (Bock and Hettenhausen 2012). DPSO was introduced in (Kennedy and Eberhart 1997) and further adapted in (Correa et al. 2006) by using a variable dimensionality for each particle. The objective function to be optimised is that of alignment quality. Particles represent candidate alignments and their dimensionality stands for the number of correspondences in an alignment. The population is predefined, e.g., 50 particles, and it evolves through a predefined number of iterations, e.g., 200, by using velocity vectors, which establish new positions of particles in the solution space based on their memory (*pBest*) and communication between them (*gBest*). Correspondence evaluation is performed through a fitness function. For this purpose, results of various matchers (applied to entities of two ontologies) are aggregated into a single fitness value. DPSO returns the global best alignment (*gBest*) with respect to the fitness function.

The method operates essentially in three steps: (i) initialisation, where each particle is assigned a random number of correspondences (since the correct number

of correspondences is not known in advance) and is evaluated through the fitness function, thus computing the $pBest$ alignments for each particle; (ii) swarm iteration, where the globally best alignment $gBest$ is updated if a new best performing particle has emerged; (iii) velocity vector update procedure, which results in a new dimensionality (updated set of correspondences) for each particle. In velocity update, in order to ensure convergence, random re-initialisations of particles are bounded. Specifically, the respective components of the velocity vectors are increased for those correspondences which are present in the global or the personal best alignments. These are controlled through the dedicated parameters: β , if a correspondence is in $pBest$, and γ , if a correspondence is in $gBest$, respectively. These correspondences are marked to be preserved (thus, not replaced by random re-initialisations in an iteration). They can be further promoted as correspondences to be never replaced in a particle with the help of customisable thresholds. For each particle its fitness and velocity vector updates can be computed in parallel.

Summary on Optimisation Techniques

We presented two different techniques for computing alignments as an optimisation problem by optimising either the quality of a similarity measure or directly that of alignments. Besides these examples, any optimisation technique may be used. The use of such methods for ontology matching is only beginning, but as soon as the problem can be reformulated as optimisation, they are usable.

We will now consider global techniques that assign a particular interpretation to the similarity or alignment that are computed. The first one assigns a probabilistic interpretation to alignments (Sect. 6.4). The second one resorts to exploiting the semantics of alignments and ontologies (Sect. 6.5).

6.4 Probabilistic Matching

Probabilistic methods may be used universally in ontology matching, e.g., to enhance some available matching candidates. In this section, we discuss several such methods, which are based on Bayesian networks (Sect. 6.4.1), Markov networks and Markov logic networks (Sect. 6.4.2). A general framework for probabilistic ontology matching is shown in Fig. 6.5.

6.4.1 Bayesian Networks

A Bayesian belief network or simply a Bayesian network is a probabilistic approach for modelling causes and effects. Bayesian networks are made up of (i) a directed

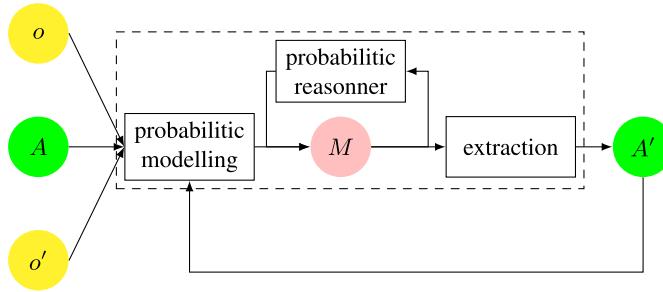


Fig. 6.5 General setting for probabilistic ontology matching. The matching problem should be first modelled within a specific probabilistic framework, e.g., as a Bayesian or Markov network. An initialisation of the model (network) is performed through a seed match computed with basic measures or by using an initial partial alignment among pairs of entities. Probabilistic reasoning is performed in the constructed model to compute an enhanced model, which can be finally extracted as an alignment.

acyclic graph, containing nodes (also called variables) and arcs, and (ii) a set of *conditional probability tables*. Arcs between nodes stand for conditional dependencies and indicate the direction of influence. For example, an arc from node X_1 (called parent) to node X_2 (called child) means that X_1 has a direct influence on X_2 . How a node influences another node (based on past experience) is defined by conditional probability tables for the nodes. $P(X|parents(X))$ is the conditional probability of variable X , where $parents(X)$ is the set of all and only nodes directly influencing X . Graph and conditional probability tables allow for constructing the joint probability distribution of all variables, namely

$$P(X_1, \dots, X_n) = \prod_i P(X_i | parents(X_i)), \quad i = 1, \dots, n$$

Given values for some nodes, it is possible to infer probability distributions for values of other nodes. In the simplest case, a Bayesian network can be specified by an expert and after some values of nodes are made observable it can be used to perform inference, thus making predictions or diagnosing causes. When not all variables are observable, it is necessary to identify dependencies between variables, which, in turn, can be solved by learning a Bayesian network that fits to the data (Russell and Norvig 1995).

Bayesian networks have been modelled and used in various ways for ontology matching. For example, two ontologies may be translated into two Bayesian networks and matching is performed as evidential inference between these Bayesian networks (Pan et al. 2005). Another work (Mitra et al. 2005) uses Bayesian networks to enhance existing matches, e.g., by deriving missed matches. Let us consider the latter in more detail.

Example 6.13 (Bayesian network, adapted from (Mitra et al. 2005)) The Bayesian network is built with correspondences and uses metarules based on the semantics of the ontology language that expresses how each correspondence affects other related correspondences. External matchers are adapted to produce initial probability distributions for correspondences, which are in turn used to infer probability distributions for other correspondences.

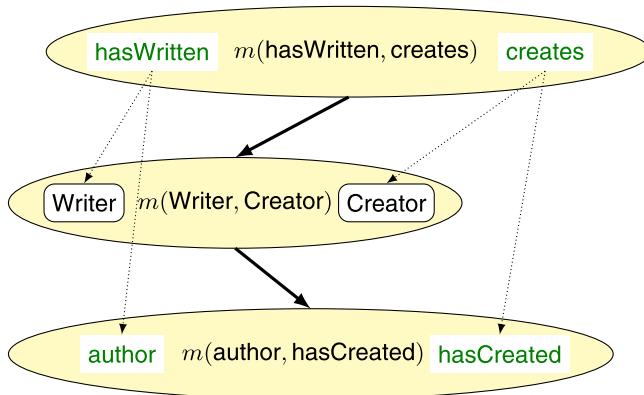


Fig. 6.6 Bayesian network graph. Each node corresponds to a correspondence between ontology entities. Dotted arrows represent relations between these entities in the ontologies which induce influence relations between correspondences represented as plain arrows.

Nodes in the Bayesian network graph represent matches between pairs of classes or properties from two distinct ontologies (see Fig. 6.6). Solid arrows in the Bayesian network graph represent the influences between its nodes, while dotted arrows stand for the relations in the ontologies under consideration. For example, the correspondence between properties $\text{hasWritten} \in o$ and $\text{creates} \in o'$ affects the correspondence between the concepts they have as range $\text{Writer} \in o$ and $\text{Creator} \in o'$, which in turn affects the correspondence between $\text{author} \in o$ and $\text{hasCreated} \in o'$. Conditional probability tables are generated by exploiting generic metarules, such as a rules stating that the probability distribution of a child node is affected depending upon the probability distribution of the parent node. For example, if two concepts Writer and Creator match and there is a relationship hasWritten between Writer and Text in o and a related relationship author between Creator and Work in o' , then we can increase the probability of match between Text in o and Work in o' . The probability distribution of the child node is derived from the probability distribution of the parent node using a set of constants. By running a Bayesian network, posterior probabilities for each node are generated.

6.4.2 Markov Networks and Markov Logic Networks

We discuss the use of both Markov networks (Pearl 1988) and Markov logic networks (Richardson and Domingos 2006), which were exploited for ontology matching in (Albagli et al. 2012) and (Niepert et al. 2010), respectively.

Markov networks are structured probabilistic networks. Similarly to Bayesian networks (Sect. 6.4.1), Markov networks, denoted as $N = \langle V, E \rangle$, are made of nodes (V) that represent variables and edges (E) between nodes that stand for statistical dependencies between the variables. Specifically, Markov networks represent joint probability distributions over events, which are represented by variables. Unlike Bayesian networks, Markov networks are *undirected*. The probability distribution in a Markov network is defined by *potential* functions (p) over the cliques (C), namely sets of nodes, such that every pair is connected by at least one edge. Potentials or tables (p_C) can be associated with each complete subgraph in the network. These correspond to conditional probabilities in Bayesian networks. The joint distribution of the event probabilities defined by Markov networks is the product of all the potentials, namely

$$P(N) = \frac{1}{Z} \prod_{C \in \text{cliques}(N)} p_C(C),$$

such that Z is a normalisation constant, also known as the partition function.

Example 6.14 (Markov network) Given a pair of ontologies, the corresponding Markov network should be built. Similarly to the example of Fig. 6.6, such a network can be built with correspondences, though the solid arrows should be substituted by undirected edges. Rules that govern the network topology could be such that ‘if two classes match, then the corresponding parent classes often match as well’. Evidence potentials can be initialised through similarity measures between all pairs of entities computed, for example, through edit distance or through user interaction. Probabilistic reasoning is performed in the constructed network, thereby computing an enhanced alignment. Inference can be performed by using Monte Carlo methods, e.g., Gibbs sampling, or through computation of posterior marginals, also known as belief propagation. Inference may be iterated or an alignment can be extracted from the network using a threshold on a posteriori probabilities.

Markov logic networks (MLN) unify Markov networks and first-order logic. In such networks, nodes represent atomic formulas and cliques represent groundings of formulas. Consider a special case of Markov networks, such as a log-linear model. In this setting, potentials are replaced by a set of features (formulas) with weights. The log-linear model probability distribution is

$$P(x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(x)\right),$$

such that Z is a normalisation constant, w_i is a real-valued weight of feature i and $f_i(x)$ is the i th feature or first-order logic formula. Thus, Markov logic networks are sets of first-order logic *formulas* (that determine graph structure) with *weights*. A grounding of a formula is obtained by replacing every variable in a formula with constants. The higher the evidence that a formula is true, the higher its weight. Thus, weights indicate how hard or soft the interpretation of a formula is intended to be. This helps in dealing with uncertainty, namely with situations that sometimes violate formulas, so such formulas become less probable, but still to a certain extent may represent a potential correspondence.

Example 6.15 (Markov logic network) Using a Markov logic network in ontology matching may consist of, given the evidence, computing the most probable alignment. The input ontologies are assumed to be true knowledge, and thus, is encoded as hard constraints that should hold in every alignment. Initial evidences or weights can be obtained by using a seed alignment between the entities to be matched. Then, a posteriori probabilities are computed for possible alignments. For example, a maximum a posteriori inference can be performed through applying integer linear programming. In this formulation of the problem, an objective function to be maximised can be as follows: $0.88f_1 + 0.1f_2 + 0.75f_3$, such that f_1 stands for the correspondence between entities such as document and documents, f_2 stands for the correspondence between provider and translator, etc. Weights represent the normalised similarity between these entities. These a priori similarities might need to exceed particular thresholds, e.g., being higher than 0.5. Integer logic program constraints, such as $f_1 + f_3 \leq 1$, are used to encode the cardinality (one-to-one), coherence (avoiding logical contradictions) or structural propagation (in the Similarity flooding spirit; see Sect. 6.2.1) constraints over alignments. The values of variables (f_i) returned through the optimal solution correspond to the final similarity.

Summary on Probabilistic Matching

As in many other domains, probabilistic modelling has found its way in ontology matching and has been fruitfully exploited. However, there remains work to be done, in particular concerning the way a priori probabilities are acquired and how the probabilistic model is built. With respect to a priori probabilities, it would certainly be more accurate to ground them on the task in which alignments are applied instead of syntactic similarities.

However, more work is needed to understand better the probabilistic foundations of ontology matching. In particular, probabilistic reasoning must be combined with the structure and semantics of alignments and ontologies. This resort to the more general topic of joining logics and probabilities and may certainly be linked to semantic techniques presented in the next section.

6.5 Semantic Techniques

The key characteristic of semantic methods is that model-theoretic semantics (Sect. 2.5.3) is used to justify their results. Hence they are deductive methods. Like other global methods, pure deductive methods do not perform very well alone for an essentially inductive task like ontology matching. They hence need to be provided with anchors (Sect. 7.3), i.e., entities which are declared, for example, to be equivalent (based on the identity of their names, external resources, or user input, for instance). These anchors constitute an initial alignment on which deductive methods can apply. Semantic methods act as amplifiers of these seed alignments.

The basis of semantic techniques is inferring new correspondences or testing the satisfiability of alignments (Sect. 2.5.3). This can be achieved by using a reasoner implementing the alignment semantics. Although there are a few such systems (Sect. 12.6), the most common technique used by far is to use the reduced semantics (Sect. 2.5.3) (Meilicke et al. 2009; Meilicke and Stuckenschmidt 2009; Meilicke 2011).

We present below semantic methods based on propositional and modal satisfiability techniques (Sect. 6.5.1), and description logic-based techniques (Sect. 6.5.2) for inferring new correspondences. Methods detecting inconsistency of alignments and repairing them are presented in the next chapter (Sect. 7.8.2).

6.5.1 Propositional Techniques

An approach for applying propositional satisfiability (SAT) techniques to ontology matching is based on the following steps (Giunchiglia and Shvaiko 2003; Shvaiko 2006):

1. Build a theory or domain knowledge (*Axioms*) for the given input ontologies as a conjunction of the available axioms. The theory is constructed by using basic techniques discussed in the previous chapter, e.g., those based on WordNet, or those using external ontologies (Sect. 7.3).
2. Build a matching formula for each pair of classes c and c' from both ontologies. The criterion for determining whether a relation holds between two classes is the fact that it is entailed by the premises (theory). Therefore, a matching query is created as a formula of the following form:

$$\text{Axioms} \Rightarrow r(c, c')$$

for each pair of classes c and c' for which we want to test the relation r (within $=, \leq, \geq, \perp$). c and c' are also sometimes called contexts.

3. Check the validity of the formula, namely that it is true for all the truth assignments of all the propositional variables occurring in it. A propositional formula is valid if and only if its negation is unsatisfiable, which is checked by using a satisfiability solver.

SAT solvers are correct and complete decision procedures for propositional satisfiability and, therefore, they can be used for an exhaustive check of all the possible correspondences. In some sense, these techniques compute the deductive closure of an initial alignment (Sect. 2.5.3).

Example 6.16 (Propositional logic relation inference)

Step 1. Assume that classes images and Europe belong to one ontology, while another ontology has classes pictures and Europe (as well). A matcher which uses WordNet can determine that images = pictures. Many other matchers can find that classes of Europe in both ontologies are identical, i.e., Europe = Europe. Then translating the relations between classes under consideration into propositional connectives in the obvious way results in the following *Axioms*:

$$(\text{images} \equiv \text{pictures}) \wedge (\text{Europe} \equiv \text{Europe})$$

Step 2. Assume that c is defined as $\text{Europe} \sqcap \text{images}$ which intuitively stands for the concept of European images, while c' is defined as $\text{pictures} \sqcap \text{Europe}$ which intuitively stands for the concept of pictures of Europe. Let us also assume that we want to know if c is equivalent (\equiv) to c' . Thus, this matching task requires constructing the following formula:

$$\begin{aligned} & ((\text{images} \equiv \text{pictures}) \wedge (\text{Europe} \equiv \text{Europe})) \Rightarrow \\ & ((\text{Europe} \wedge \text{images}) \equiv (\text{Europe} \wedge \text{pictures})) \end{aligned}$$

Step 3. Negation of this formula turns out to be unsatisfiable, and therefore, the equivalence relation holds. See Sect. 11.3 for a detailed discussion of this example.

This technique, besides pruning incorrect correspondences, also infers new ones between complex concepts. In the example above, c is defined by combining (taking the intersection of) such atomic concepts as Europe and images. And, similarly for c' . These are simple examples of complex concepts within the expressive power of a propositional language. The relation between such complex concepts as $(\text{Europe} \wedge \text{images})$ and $(\text{Europe} \wedge \text{pictures})$ was not available after the first step, and has been deduced by the reasoner.

This technique can only be used for matching tree-like structures, such as classifications, taxonomies, without taking properties or roles into account. Modal SAT can be used, as proposed in (Shvaiko 2006), for extending the methods related to propositional SAT to binary predicates.

6.5.2 Description Logic Techniques

In description logics, the alignment relations, e.g., $=$, \leq , \geq , \perp , can be expressed with respect to subsumption. The subsumption test, can be used to establish the

relations between classes in a purely semantic manner. In fact, first merging two ontologies (after renaming) and then testing each pair of concepts and roles for subsumption is enough for matching terms with the same interpretation (or with a subset of the interpretations of the others) (Bouquet et al. 2006).

Example 6.17 (Description logic relation inference) Consider two minimal description logic ontologies:

$$\text{Micro-company} \equiv \text{Company} \sqcap \leq_5 \text{employee}$$

meaning that a Micro-company is a Company with at most 5 employees and

$$\text{SME} \equiv \text{Firm} \sqcap \leq_{10} \text{associate}$$

meaning that a SME is a Firm with at most 10 associates. The following initial alignment (expressed in description logic syntax) includes:

$$\text{Company} \equiv \text{Firm}$$

$$\text{associate} \sqsubseteq \text{employee}$$

It expresses that Company is equivalent to Firm and associate is a subproperty of employee. This obviously entails:

$$\text{Micro-company} \sqsubseteq \text{SME}$$

i.e., Micro-company is a subclass of SME.

ContentMap (Sect. 8.4.7) is an interactive tool for diagnosing and repairing alignments. From an alignment, it generates consequences of the merged OWL ontologies using a reasoner (Pellet). It then presents them to users who can then choose the undesirable consequences. Finally, ContentMap provides a repair plan, i.e., a set of correspondences that will be suppressed from the alignment.

There are other uses of description logic techniques which are relevant to ontology matching. For example, in a spatio-temporal database integration scenario, as first motivated in (Parent and Spaccapietra 2000) and later developed in (Sotnykova et al. 2005), the inter-schema correspondences are initially proposed by the integrated schema designer and are encoded together with input schemas in the $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ language. Then, description logic reasoning services are used to check the satisfiability of the two source schemas and the set of inter-schema correspondences. If some objects are found unsatisfiable, the inter-schema correspondences should be reconsidered.

Summary on Semantic Techniques

Semantic techniques are invaluable when correspondences are generated in order to ensure the completeness, i.e., find all the correspondences that must hold (Sect. 6.5.1)

and Sect. 6.5.2), and the consistency, i.e., find correspondences that lead to inconsistency (Sect. 7.8.2), of the alignment. The two types of operations can naturally be used together. Only a few of these techniques have been developed so far. However, with the improvement of deductive tools for dealing with semantic web languages, we expect more systems using semantic-based techniques. Semantic techniques are a good starting base to the development of a more general approach to revision and update in alignments (Qi et al. 2009) and networks of ontologies.

6.6 Summary

Contrasting with basic matchers presented in Chap. 5, we have discussed approaches that compare ontologies and ontology entities globally. Such techniques take advantage of basic matchers to provide them anchor or seed alignments, but require specific principles to propagate comparisons globally. These may be based on structural relations, semantic interpretations or probabilities.

As a result, the presented techniques can provide similarities or alignments. These may be combined with other similarities and alignments or manipulated like the result of any other matcher. Chapter 7 explores the various techniques for doing this and discusses how to compose matchers into a coherent system.

Chapter 7

Matching Strategies

The basic techniques presented in Chap. 5 and the global techniques provided in Chap. 6 are the building blocks on which a matching system is built. Once the similarity or dissimilarity between ontology entities is available, the alignment remains to be computed. This involves more comprehensive treatments. In particular, the following aspects of building a working matching system are considered in this chapter:

- preparing, if necessary, to handle large scale ontologies (Sect. 7.1),
- organising the combination of various similarities or matching algorithms (Sect. 7.2),
- exploiting background knowledge sources (Sect. 7.3),
- aggregating the results of the basic methods in order to compute the compound similarity between entities (Sect. 7.4),
- learning matchers from data (Sect. 7.5) and tuning them (Sect. 7.6),
- extracting alignments from the resulting (dis)similarity: indeed, different alignments with different characteristics may be extracted from the same (dis)similarity (Sect. 7.7),
- improving alignments through disambiguation, debugging and repair (Sect. 7.8).

7.1 Ontology Partitioning and Search-Space Pruning

Matchers may have to deal with large ontologies containing tens to hundreds of thousands of entities each. In order to do this efficiently, it is possible either to split the ontologies into smaller ontologies, this is called partitioning, and to match these smaller ontologies (Sect. 7.1.1), or to dynamically ignore parts of the ontologies when matching, this is called pruning (Sect. 7.1.2).

7.1.1 Partitioning

It is often useful to circumscribe parts of the ontologies that have to be matched, to perform matching by parts and to aggregate the results of independent matching (usually by simply accepting all the generated correspondences). This is presented in Fig. 7.1.

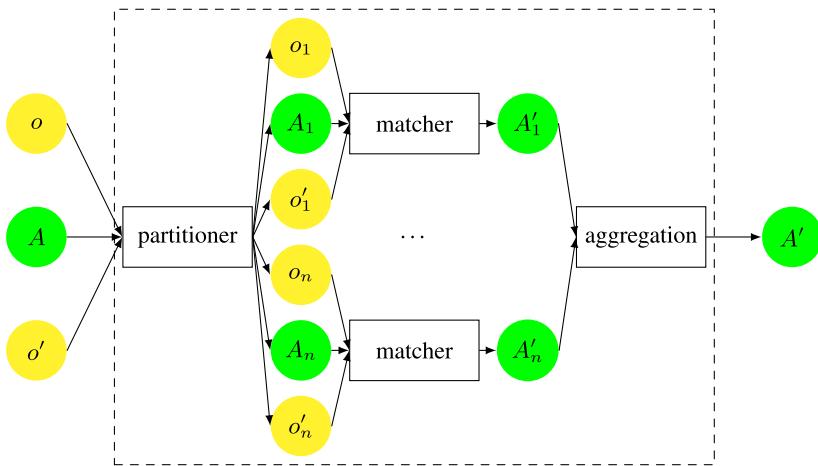


Fig. 7.1 General principle of partitioning: the ontologies to match are split into smaller ontology fragments which can be matched independently and their results merged.

We refer to this process as ontology partitioning, though this may not be a strict mathematical partition, i.e., dividing the set of entities into jointly exhaustive and non-overlapping classes. Indeed, some parts of an ontology may be simply ignored because these do not have any counterparts in the other ontology and some parts can be tentatively matched with several other counterparts.

Figure 7.1 is simplistic because some systems may perform an iterated partitioning, for instance, by starting with the top-level entities and then by partitioning iteratively subclasses based on the obtained result. This may also work from leaf entities to the root.

In other contexts, this process is called blocking, i.e., partitioning the ontologies into blocks, avoiding the Cartesian product of comparisons among the ontologies in favour of matching entities only from corresponding blocks. Technically, blocking is performed through matching the entities to a multidimensional index, in which similar entities are positioned near each other in the index. Blocking is typically used in data interlinking due to the large number of entities to consider.

Partitioning and pruning techniques often use anchors. A relatively fast technique for obtaining anchors consists of finding entities with exactly matching nodes for some hashing function. For that purpose, instead of comparing all nodes (complexity

$O(n^2)$) it is possible to build an index of all the entities of one ontology, typically through a hash-table, and to check for each entity of the other ontology, there is a corresponding indexed term (complexity: $O(2n)$).

Partitioning may use techniques developed for ontology modularisation (Stuckenschmidt et al. 2009) (and vice versa (Ghazvinian et al. 2011)), but ontologies are partitioned with respect to each other, instead of being partitioned independently. For example, Falcon-AO (Sect. 8.3.9) clusters ontologies in blocks using the ROCK agglomerative clustering algorithm (Guha et al. 1999). The algorithm takes into account two criteria for designing blocks: the internal cohesion of a block and the pairing of a block with another block in the other ontology. These are summarised into a single measure of *goodness*, such that given two sets of classes (blocks) B and B' :

$$\begin{aligned} \text{cohesion}(B) &= \text{goodness}(B, B) \\ \text{pairability}(B, B') &= \text{goodness}(B, B') \\ \text{goodness}(B, B') &= \frac{\sum_{c \in B, c' \in B'} \text{link}(c, c')}{|B| \times |B'|} \end{aligned}$$

such that *link* is a normalised similarity measure between two classes that can be obtained quickly. Then Falcon-AO pairs the blocks to be compared using a weighted sum of a lexical similarity and the Wu–Palmer similarity. The choice of a pair of blocks to be matched is based on the availability of anchors between them. These anchors are compared to anchors with other blocks. If this measure is superior to a predefined threshold, then the blocks will be matched.

TaxoMap (Sect. 8.1.46) offers two partitioning methods refining the Falcon-AO method by attempting to introduce more dependency between the two partitions (Hamdi et al. 2010b). The first algorithm, Partition-Anchor-Partition, first partitions one of the ontologies based on its own structure; then it computes anchors between the ontologies and partitions the second ontology starting at the anchors. The second method, Anchor-Partition-Partition, is more adapted to unstructured ontologies. It partitions the two ontologies by starting from the anchors that are found in these ontologies, and for the second ontology, groups of anchors that are in the same blocks. In both cases, blocks are then paired together based on anchors. (Doran et al. 2009) has studied the benefits of different partitioning strategies for reducing the matching search space.

Partitioning can be used for parallelising a matcher in a simple paradigm such as MapReduce (Dean and Ghemawat 2004; Lin and Dyer 2010). Indeed, each node could operate the map operation by (i) partitioning the problem, (ii) submitting each subproblem to nodes as soon as they are available, and (iii) aggregating the resulting subalignments together, and perform the reduce operation by simply matching two ontologies and returning an alignment. The difficulty consists of finding a partition that does not alter the quality of the alignment by not providing enough information to matchers. Conversely, redundant information can also be removed in order to

use less memory. The subproblems identified can also be persistently cached. In turn, this cache may be replicated over nodes in a cloud, thus facilitating loading of matching tasks in parallel.

7.1.2 Search-Space Pruning

Pruning techniques dynamically avoid comparing parts of the ontologies without partitioning them beforehand.

AROMA (Sect. 8.3.14) learns association rules from the more generic to the more specific. At each stage, it can measure the maximum implication intensity that can be obtained by learning more specific rules. If this value is under a threshold, more specific rules will not be considered, avoiding many comparisons (David et al. 2007). Then, it only investigates further correspondences between two pairs of anchors given either by the previous techniques or by the previous fast equality index.

Lily (Sect. 8.1.41) defines reduction anchors, which dynamically decide which comparisons will be avoided (Wang et al. 2011). When two classes are put in a correspondence, positive anchors prohibit the comparison of the subclasses of one with the superclasses of the other; negative anchors prohibit the comparison of the neighbourhood of a class with the neighbourhood of a class it does not match (or for which the similarity is low).

Anchor-Flood (Sect. 8.1.40) also starts with anchors in the same way as above. It then compares the neighbourhoods (or segments) of both anchors, i.e., the set of entities connected to the anchors two levels away (parents, grandparents, children, grandchildren, siblings, etc.). The algorithm only compares entities from two such anchored segments, starting from the anchors and spreading to the neighbourhood until all entities are reached (Hanif and Aono 2009). The pairs of entities to match are those reachable from the same type of operation (ascending, descending, sibling).

LogMap (Sect. 8.3.26) adopts a dual strategy. It first indexes all the entities on the basis of their labels and URIs. It uses terminological techniques, such as stemming or lexicons (Sect. 5.2.2), in order to increase recall. It extracts candidate correspondences from each pair of entities indexed together. Only these candidate correspondences will be considered for matching, hence the pruning should be strong enough for limiting comparisons, but weak enough for not restricting too much the scope of the resulting alignments. In addition, because LogMap uses semantic methods (Sect. 6.5), it also partitions the ontologies so as to only use ontology modules relevant to each candidate correspondence to consider. Similarly, PORSCHE (Sect. 8.1.38) and XClust (Sect. 8.1.15) first cluster entities together in order to reduce the target search space for a source ontology entity.

7.2 Matcher Composition

All the steps mentioned above are considered under the name of global methods. The goal of a global method is to combine local methods (or basic matchers) in order to define a new matching algorithm. We present here, at the strategic level, some natural ways to combine matchers. For that purpose, we progressively introduce new graphical elements. These are summarised in Fig. A.3 (Appendix A).

So far, we have only presented the outside of the matching process by producing an alignment from two ontologies such as in Fig. 2.8. A natural way of composing basic matchers consists of improving the alignment through the use of sequential composition (see Fig. 7.2). For instance, one would like to first use a matcher based on labels (Sect. 5.2) before running another one based on the structure of entities (Sect. 5.3) or a semantic matcher (Sect. 6.5).

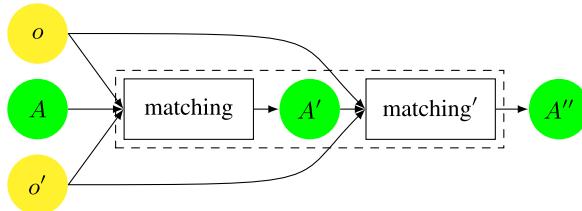


Fig. 7.2 Sequential composition of matchers.

This sequential process can be used, for instance, in on-line data integration. Ontology matching and integration consists of merging data (and sometimes data streams, d and d') expressed in different ontologies (o and o'). For that purpose, the ontologies have to be matched beforehand and the data integration can use this alignment. This is an example of combined off-line and on-line matching.

It can be thought of as (see Fig. 7.3):

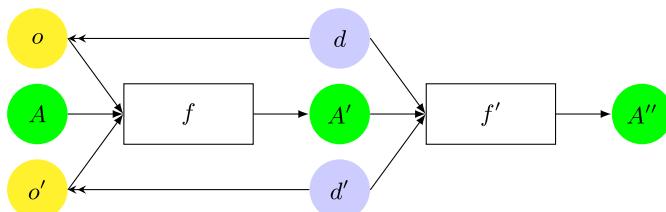


Fig. 7.3 Instance matching as another matching process taking advantage of a prior matching of ontologies (o and o') for integrating data flows (d and d'). This is typically what is exploited in Fig. 1.5 (data interlinking).

1. a first matching phase (f), possibly with an instance training set,
2. a data matching phase (f') using the first alignment (A').

In this setting, the second phase benefits from the precompiling of the first alignment. Indeed, the second matcher f' can be thought of as a compilation of the first alignment.

However, the sequential combination of matchers is more classically used to improve an alignment. For that purpose, when using similarities or distances, the matchers can be sequentially composed through their similarity matrix. We introduce, in Fig. 7.4, new symbols for matrices as well as a new component for extracting an initial matrix from either an initial alignment or a pair of ontologies (first triangle) and another one for extracting an alignment from a similarity or dissimilarity matrix (sparrow tailed triangle, detailed in Sect. 7.7).

The sequential composition through a distance or similarity matrix is illustrated in Fig. 7.5.

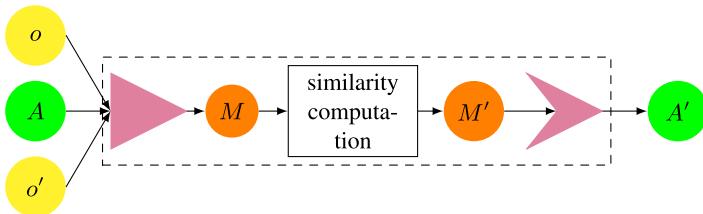


Fig. 7.4 Introduction of a (virtual) matrix which represents a similarity or distance measure between entities to be matched. The first operator builds an initial matrix M from the two ontologies o and o' . The core of the matching algorithm produces a similarity or distance matrix M' from this initial matrix and the description of the ontologies. Finally, alignment A' is extracted from matrix M' .

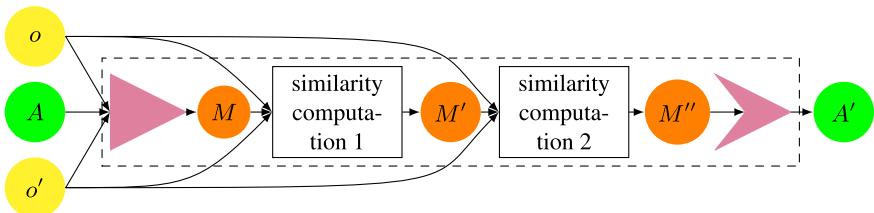
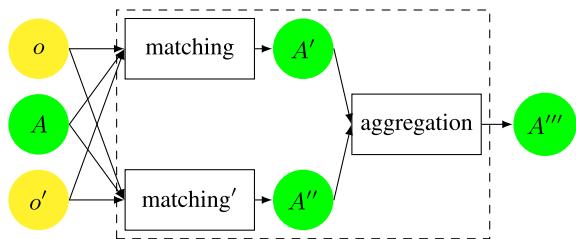


Fig. 7.5 Sequential composition of matchers through similarity.

Another way to combine algorithms consists of running several different algorithms independently and aggregating their results (see Fig. 7.6): this is called parallel composition. Such aggregation techniques may be very different: it may correspond to choosing one of the results on some criterion or merging their results

through some operator. For instance, it can consist of running several matching algorithms in parallel and selecting the correspondences which are in all of them (intersection is then used as an aggregation operator) or selecting all the correspondences with highest confidence.

Fig. 7.6 Parallel composition of matchers.



In the latter case, it is often more convenient to define the aggregation operators on the similarity or distance matrix (see Fig. 7.7) because there are many mathematical techniques available for that purpose. These techniques are presented in Sect. 7.4.

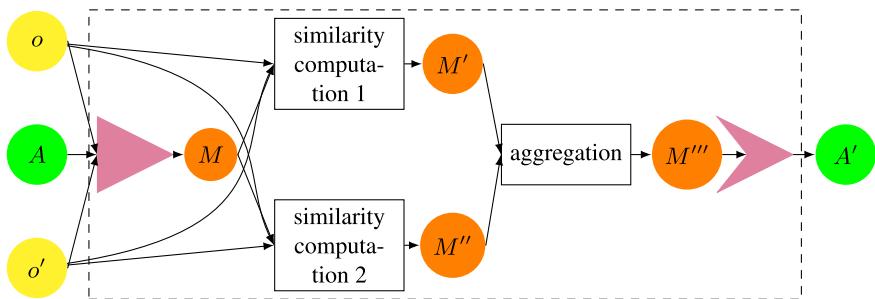


Fig. 7.7 Parallel composition of matchers through similarity.

There are two main kinds of parallel composition:

Heterogeneous parallel composition in which the input is fragmented into different kinds of data (graphs, strings, sets of documents, etc.) and the aggregation takes advantage of all of them (by aggregating their results) or the most promising only. This is the topic of Sect. 7.4.

Homogeneous parallel composition in which the input goes into several competing systems and the aggregation selects the best of these or some consensus between them.

Of course, it is possible to combine these two classes even further.

All these composition techniques are usually implemented within particular matching algorithms (which are presented in Chap. 8). However, there are some

systems that offer the opportunity to combine other systems, such as FOAM (Sect. 10.3.7), Rondo (Sect. 10.3.1) or the Alignment API (Sect. 10.3.6).

7.3 Context-Based Matching

When two ontologies have to be matched, they often lack a common ground on which comparisons can be based. The goal of ontology matching is to find this ground. This may be achieved by comparing the content of the ontology or by dealing with its context, i.e., the relations that the ontology has with the environment in which it is used.

This common ground can often be found by relating the ontologies to external resources. These resources may differ on three specific dimensions:

Breadth: whether they are general-purpose resources or domain-specific resources.

By using specialised resources, e.g., the Foundational Model of Anatomy in medicine, one can be sure that the concepts in the contextualised resources can be matched accurately to their corresponding concepts in the ontology. However, by using more general resources there is more probability that an alignment already exists and can be exploited right away.

Formality: whether they are pure ontologies defined in a formal language (the external resource is then called background knowledge (Giunchiglia et al. 2006c)), less formal resources such as WordNet or fully informal resources such as Wikipedia. By using formal resources, e.g., DOLCE or FMA, it is possible to reason within or across these formal models in order to deduce the relation between two terms. By using terminological resources, e.g., WordNet (Sect. 5.2.2), it is possible to extend the set of senses that are covered by a term and to increase the number of terms that can express these concepts. There is thus more opportunities to match terms.

Status: whether these resources are considered as references such as ontologies, thesauri or they are sets of instances or annotated documents that are shared (see Sect. 5.4.1).

Contextualising ontologies can typically be achieved by matching these ontologies with a common upper-level ontology (Sect. 2.1.6) that is used as an external source of common knowledge.

Example 7.1 (Using upper-level ontologies as background knowledge) An experiment was carried out by expressing fishery resources (such as databases and thesauri) within the DOLCE upper-level ontology (Gangemi 2004). The goal was to merge these resources into a common Core Ontology of Fisheries. It involved transforming manually the resources into lightweight ontologies expressed with respect to DOLCE and then using reasoning facilities for detecting relations and disjointness between entities of this ontology.

Example 7.2 (Using domain-specific formal ontologies as background knowledge) Assume that the anatomy part of the CRISP directory¹ has to be matched to the anatomy part of the MeSH² metathesaurus. In this case, the FMA ontology can be used as background knowledge, which gives the context to the matching task. The result of anchoring is a set of matches with three different kinds of relations: $=$, \leq , \geq between concepts from FMA, and CRISP or MeSH.

For example, the concept of brain from CRISP, denoted by $\text{Brain}_{\text{CRISP}}$, could be easily anchored to the concept of brain in FMA, denoted by $\text{Brain}_{\text{FMA}}$. Similarly, the concept of head from MeSH, denoted by $\text{Head}_{\text{MeSH}}$, could be anchored to a background knowledge concept Head_{FMA} . In the reference ontology FMA there is a *part of* relation between $\text{Brain}_{\text{FMA}}$ and Head_{FMA} . Therefore, we can derive that $\text{Brain}_{\text{CRISP}}$ is a part of $\text{Head}_{\text{MeSH}}$.

Since the domain-specific ontology provides the context for the matching task, the concept of Head was correctly interpreted as meaning the upper part of the human body, instead of, for example, meaning a chief person. This is not so straightforward, as can be shown by replacing FMA with WordNet: in WordNet the concept of Head has 33 senses (as a noun). Finally, once the context of the matching task has been established, as the example shows, various heuristics, such as string-based techniques, can improve the anchoring step.

Because context-based matching is very versatile, we synthesise its behaviour in a generalised view that aims at covering and extending existing matchers, such as Scarlet (Sect. 8.1.32) or OMviaUO (Sect. 8.1.33). This framework describes the use of formal ontologies as background knowledge and can be adapted to deal with informal resources or linked data for instance. For that purpose, we decompose context-based matching into seven steps described in Fig. 7.8:

Ontology arrangement preselects and ranks the ontologies to be explored as intermediate ontologies. The preselection may retain all the ontologies from the web or ontologies belonging to a particular type, such as upper-level ontologies, domain-specific ontologies, e.g., medical or biological ontologies, competencies, popular ontologies, recommended ontologies, or any customised set of ontologies.

The ordering may be based on the likeliness of the ontology being useful, usually measured as a distance. Such a distance may be based on the proximity of the ontology with the ontology to be matched (David and Euzenat 2008), the existence of alignments between them (David et al. 2010), or the availability of quickly computable anchors (Sect. 7.1.1).

Contextualisation (or **anchoring**) finds anchors between the ontologies to be matched and the candidate intermediate ontologies. These anchors can be correspondences of any type, including various relations and confidence measures. In principle, any ontology matching method presented in this book can be used

¹Now replaced by the RePorter system: <http://projectreporter.nih.gov/>.

²<http://www.nlm.nih.gov/mesh/>.

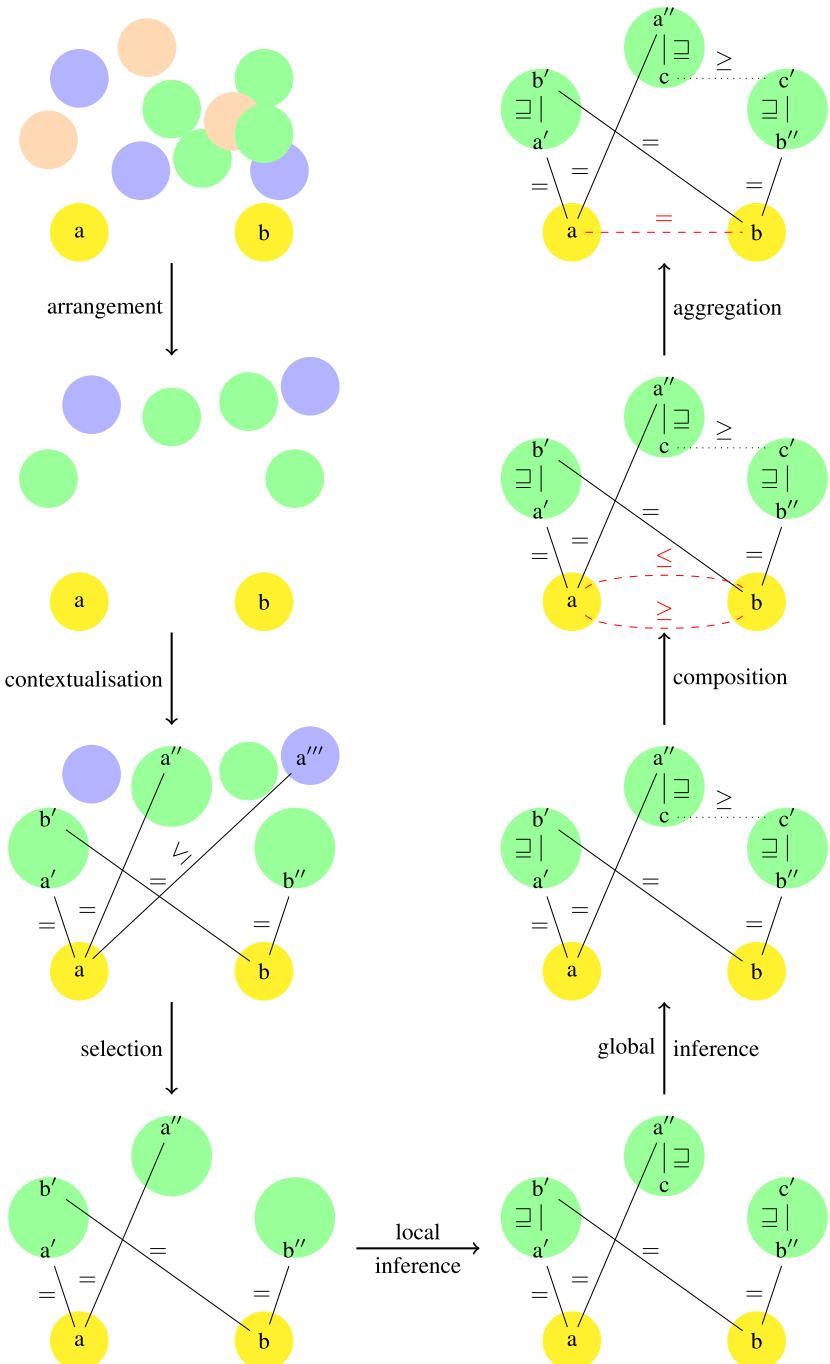


Fig. 7.8 The different steps of context-based matching (from (Locoro et al. 2013)).

for anchoring; in practice, this is usually a fast method, like string matching (see Sect. 7.1.1), because anchoring is only a preliminary step. In Fig. 7.8, a is contextualised as a' , a'' and a''' ; b is contextualised as b' and b'' .

Ontology selection restricts the candidate ontologies that will actually be used.

This selection relies usually on the computed anchors by selecting those ontologies in which anchors are present. In Fig. 7.8, ontologies with no anchors and blue ontologies with only one anchor are eliminated.

Local inference obtains relations between entities of a single ontology. It may be reduced to logical entailment. It may also use weaker procedures, especially when intermediate resources have no formal semantics, e.g., thesauri. It could then be replaced by the use of asserted relations of the ontologies or relations obtained through composing existing ones. In Fig. 7.8, b' subsumes a' . Other relations, such as $a'' \sqsupseteq c$ and $c' \sqsupseteq b''$, were inferred as well.

Global inference finds relations between two concepts of the ontologies to be matched by concatenating relations obtained from local inference and correspondences across intermediate ontologies. In Fig. 7.8, from the former assertions and a new correspondence $c \geq c'$, it is possible to infer $a'' \geq b''$.

Composition determines the relations holding between the source and target entities by composing the relations in the path (sequence of relations) connecting them. The composition method can be functional ($= \cdot =$ is $=$), order-based ($< \cdot \leq <$) or relational ($\perp \cdot \geq \perp$ is \perp). In Fig. 7.8, there exist two paths supporting composition: $a = a' \sqsubseteq b' = b$ and $a = a'' \sqsupseteq c \geq c' \sqsupseteq b'' = b$. They yield the following assertions: $a \leq b$ and $a \geq b$.

Aggregation combines relations obtained between the same pair of entities. It can either simply return all the correspondences or return only one correspondence with an aggregated relation. Aggregation itself can be based on various methods, such as relation aggregation operators, e.g., conjunction, popularity (selecting the relation which is obtained from the most paths) or confidence (selecting the relation with the highest confidence). In Fig. 7.8, the two former relations are aggregated by conjunction as $a = b$.

These steps extend those provided by Scarlet (Sect. 8.1.32): contextualisation was called anchoring, selection was considered, local and global inference as well as composition were gathered in a set of ‘derivation rules’ and aggregation was called combining. GeRoMeSuite (Sect. 8.1.30) also identified the arrangement (called selection), anchoring, local inference (including composition), and aggregation steps to which a consistency check is added. This presentation provides a finer-grained decomposition of context-based matching that can be used for instantiating differently each step.

Context-based matching may be seen from a fully logical point of view: local and global inference are replaced by entailment tests and composition and aggregation are replaced by logical deduction. In such a case, beyond anchoring, matching is reduced to reasoning in a network of ontologies. Hence, when the technology for reasoning in networks of ontologies will be fully developed, it will be

possible, in principle, to reduce the seven steps above to anchoring and reasoning (see Sect. 6.5).

The difficulty of context-based matching is a matter of balance: adding context provides new information, and hence, helps increase recall, but this new information may also generate incorrect correspondences that decrease precision. Many options can be exercised concerning the type of resource to be used or the way it is connected to the ontologies to be matched.

7.4 Similarity and Alignment Aggregation

As Sect. 7.2 showed, the composition of matchers can be achieved by aggregating similarities. This aggregation takes similarities provided by different matchers and combines them into a single similarity. Matchers can be competing, i.e., they match the same types of entities with different assessments of the respective similarities, or complementary, i.e., they identify similarities in different types of entities. These will have to be aggregated differently. Here we identify three different types of aggregations: weighting (Sect. 7.4.1), which arithmetically combines similarities by giving different weights to matchers, voting (Sect. 7.4.2), which makes matchers vote on entity similarity or correspondences, and arguing (Sect. 7.4.3), which considers matcher results as arguments for or against correspondences and uses argumentation techniques for selecting correspondences.

7.4.1 Weighting

Compound similarity is concerned with the aggregation of heterogeneous similarities. As explained in Sect. 5.3, structured objects (classes, individuals) are very often involved in many different relations, so it is possible to compute a similarity between each of the ontology entities two objects are related with. For instance, the similarity between two classes may rely on the similarity obtained from their names, the similarity of their superclasses, the similarity of their instances and that of their properties. These similarities have to be aggregated, in a single similarity measure.

Triangular Norms

Triangular norms are used as conjunction operators in uncertain calculi.

Definition 7.3 (Triangular norm) A triangular norm T is a function from $D \times D \rightarrow D$ (such that D is a set ordered by \leq and provided with an upper bound T)

satisfying the following conditions:

$$\begin{aligned} T(x, \top) &= x && \text{(boundary condition)} \\ x \leq y \Rightarrow T(x, z) &\leq T(y, z) && \text{(monotony)} \\ T(x, y) &= T(y, x) && \text{(commutativity)} \\ T(x, T(y, z)) &= T(T(x, y), z) && \text{(associativity)} \end{aligned}$$

Typical examples of triangular norms are $\min(x, y)$, $x \times y$ and $\max(x + y - 1, 0)$. All are normalised if the measures provided to them are normalised; \min is the only idempotent norm ($\forall x, \min(x, x) = x$). Triangular norms are the obvious candidates for a combination that requires the highest score from all aggregated values. Due to associativity, triangular norms can be extended to n -ary measures. Any triangular norm over the unit interval can be expressed as a combination of these three functions (Hájek 1998).

Another triangular norm for aggregating several dimensions is the weighted product.

Definition 7.4 (Weighted product) Let o be a set of objects that can be analysed in n dimensions, the weighted product between two such objects is $\forall x, x' \in o$,

$$\delta(x, x') = \prod_{i=1}^n \delta(x_i, x'_i)^{w_i}$$

such that $\delta(x_i, x'_i)$ is the dissimilarity of the pair of objects along the i th dimension and w_i is the weight of dimension i .

These operators have the drawback that if one of the dimensions has a measure of 0, then the result is also 0.

Example 7.5 (Triangular norms) We consider in this section two ontologies comprising the concepts Product, Provider, Creator for the first one and Book, Translator, Publisher and Writer for the second one.

The two tables below display the result of applying an edit distance and a WordNet-based distance on these labels.

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.86	.8	.89	.86	Product	.82	.88	.88	.85
Provider	.88	.8	.56	.5	Provider	.83	.89	.76	.71
Creator	.86	.5	.89	.57	Creator	.82	.53	.88	.85
Normalised Levenshtein distance					Alignment API WordNet-based distance				

The following tables display the aggregations of these distances with triangular norms, namely, the \min operation and a weighted product.

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.82	.8	.88	.85	Product	.84	.84	.88	.85
Provider	.83	.8	.56	.5	Provider	.85	.84	.65	.60
Creator	.82	.5	.88	.57	Creator	.84	.51	.88	.70
Minimum of the distances					Weighted product with $w_1 = w_2 = \frac{1}{2}$				

Since the two first similarities were not very dissimilar from each other, the results of the two operators are very similar as well.

Contrary to the multidimensional aggregators, triangular norms tend to imply dependencies between the values of the different dimensions, so that the value given on one dimension can override a value on another dimension.

Multidimensional Distances and Weighted Sums

In case the difference between some properties must be aggregated, one of the most common family of distances are the Minkowski distances. Contrary to the previous ones, these measures are well suited to independent dimensions and tend to balance the values between dimensions.

Definition 7.6 (Minkowski distance) Let o be a set of objects that can be analysed in n dimensions, the Minkowski distance between two such objects is $\forall x, x' \in o$,

$$\delta(x, x') = \sqrt[p]{\sum_{i=1}^n \delta(x_i, x'_i)^p}$$

such that $\delta(x_i, x'_i)$ is the dissimilarity of the pair of objects along the i th dimension.

Instances of the Minkowski distances are the Euclidean distance (when $p = 2$), the Manhattan, a.k.a. City-blocks, distance (when $p = 1$) and the Chebichev distance (when $p = +\infty$). These should be used when aggregating measures from independent dimensions.

Example 7.7 (Minkowski distances) We start with the distance computed on labels with the min aggregation operator in Example 7.5 and a distance obtained from the Hamming distance on the set of instances of concepts. These distances typically take into account independent dimensions.

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.82	.8	.88	.85	Product	.8	1.	1.	1.
Provider	.83	.8	.56	.5	Provider	1.	1.	.15	.98
Creator	.82	.5	.88	.57	Creator	1.	.83	.99	.22

Minimum of the distances

Distances obtained by using the Hamming distance on sets of the concept instances. The relatively high distance between Product and Book is due to the large number of Products which are not Books.

The aggregation of these two distances using (normalised) Euclidean and Manhattan distances are as follows:

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.86	.96	1.	.99	Product	.86	.96	1.	.98
Provider	.98	.96	.44	.83	Provider	.97	.96	.38	.79
Creator	.97	.73	1.	.46	Creator	.97	.71	.99	.42

Normalised Euclidean distance based on the two above dimensions

Normalised Manhattan distance based on the two above dimensions

The values given by the Euclidean distance are lower than those of the Manhattan distance, though they are very close.

These distances can be weighted in order to give more importance to some dimensions. They can be normalised by dividing their results by the maximum possible distance (which is not always possible) but they have the main drawback of not being linear if $p \neq 1$. This is a source of problems when trying to find these distances if they are defined as functions of each others (see Sect. 6.2 and (Valtchev 1999)).

A simple linear aggregation can be further refined by adding weights to this sum. Weighted linear aggregation considers that the values to be aggregated do not have the same importance. For instance, similarity in properties is more important than similarity in comments. The aggregation function will thus use a set of weights w_1, \dots, w_n corresponding to a category of entities, e.g., classes, properties. The aggregation function can be defined as follows:

Definition 7.8 (Weighted sum) Let o be a set of objects that can be analysed in n dimensions, the weighted sum between two such objects is $\forall x, x' \in o$,

$$\delta(x, x') = \sum_{i=1}^n w_i \times \delta(x_i, x'_i)$$

such that $\delta(x_i, x'_i)$ is the dissimilarity of the pair of objects along the i th dimension and w_i is the weight of dimension i .

The weighted sum can be thought of as a generalisation of the Manhattan distance in which each dimension is weighted. It also corresponds to weighted average with normalised weights. In fact, the weights can be different depending on the categories of the objects aggregated (Sect. 6.2.2). Then, the function can use a set of weights w_C^P depending on the category of object C and the kind of value computed P .

This kind of measures can be normalised, if all values are normalised, by having $\sum_{i=1}^n w_i = 1$.

Example 7.9 Weighted sum From Example 7.7, it appears that the measure on the instances is more accurate than those on the labels. This can be inferred from the fact that there are no common names in both sets of labels or that there are lower distances in the latter case. Thus, weighting these dimensions could be promising. Let us consider the same input set as in Example 7.7. The computed weighted sums are as follows:

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.81	.93	.96	.95	Product	.81	.95	.97	.96
Provider	.94	.93	.29	.82	Provider	.96	.95	.25	.86
Creator	.94	.72	.95	.34	Creator	.96	.75	.96	.31
Normalised weighted sum with $w_{label} = 1/3$ and $w_{inst} = 2/3$					Normalised weighted sum with $w_{label} = 1/4$ and $w_{inst} = 3/4$				

The results clearly identify $\langle \text{Provider}, \text{Publisher} \rangle$ and $\langle \text{Creator}, \text{Writer} \rangle$ as candidate matches. The low similarity between Product and Book prevents from choosing them as a match candidate.

Fuzzy Aggregation and Weighted Average

Fuzzy aggregation operators are used for assimilating homogeneous quantities in a way that preserves the structure of the aggregated domains.

Definition 7.10 (Fuzzy aggregation operator) A fuzzy aggregation operator f is a function from $D^n \rightarrow D$ (with D being a set ordered by \leq and provided with an upper bound \top) satisfying $\forall x, x_1, \dots, x_n, y_1, \dots, y_n \in D$ the following conditions:

$$f(x, \dots, x) = x \quad (\text{idempotency})$$

$$\forall x_i, \quad y_i, x_i \leq y_i \quad \Rightarrow \quad f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n) \quad (\text{increasing monotony})$$

$$f \text{ is a continuous function} \quad (\text{continuity})$$

\min is also a fuzzy aggregation function. A general result about these measures is that for any fuzzy aggregation function f , the aggregation is ordered by $f(x, y) \geq \min(x, y) \geq x \times y \geq \max(x + y - 1, 0)$. A typical example of a fuzzy aggregation operator is the weighted average (Gal 2011).

Definition 7.11 (Weighted average) Let o be a set of objects that can be analysed in n dimensions, the weighted average between two such objects is $\forall x, x' \in o$,

$$\delta(x, x') = \frac{\sum_{i=1}^n w_i \times \delta(x_i, x'_i)}{\sum_{i=1}^n w_i}$$

such that $\delta(x_i, x'_i)$ is the dissimilarity of the pair of objects along the i th dimension and w_i is the weight of dimension i .

A simple average function is a function such that all weights are equal. If the values are normalised, the weighted average is normalised. In fact, the normalised weighted sum is also a weighted average (see Example 7.9).

Fuzzy aggregation functions have to be used when aggregating the results of competing algorithms (which are efficient with respect to some aspects and not with respect to others) and trying to take advantage of all of them. They are very useful if one wants to use a learning algorithm for learning the weights of the measure (see Sect. 7.5). (Gal et al. 2005a) argues that these measures are always preferable to triangular norms for aggregating confidence measures.

Some systems, such as LCS (Sect. 8.4.2) and MoTo (Sect. 8.3.24), took inspiration from linguistic quantifiers (Yager 1988, 1993), e.g., ‘most of’, to define fuzzy aggregation functions.

Harmonic Adaptive Weighted Sum

(Mao et al. 2010) introduced the notion of harmonic adaptive weighted sum for weighting different matchers. This operation gives a higher weight to measures that are more discriminant. For each measure, the operator computes the ratio of cells in the matrix whose similarity is inferior (for a dissimilarity) to all the other cells in the same row and column over the maximum possible such value (the size of the smallest ontology).

Definition 7.12 (Harmonic adaptive weighted sum) Let o and o' be two sets of objects that can be compared by n measures, the harmonic adaptive weighted sum of such measures between two such objects is $\forall x \in o, x' \in o'$,

$$\delta(x, x') = \sum_{i=1}^n h(\delta_i) \times \delta_i(x, x')$$

such that $\delta_i(x, x')$ is the measure of dissimilarity of the pair of objects along the i th measure and h is such that

$$h(\delta_i) = \frac{|\{ \langle e, e' \rangle \in o \times o'; \wedge \begin{array}{l} \forall y \in o' \setminus \{e'\}, \sigma(e, y) > \sigma(e, e') \\ \forall x \in o \setminus \{e\}, \sigma(x, e') > \sigma(e, e') \end{array} \}|}{\min(|o|, |o'|)}$$

This measure can be normalised by dividing the results by the sum of the weights. It favours those similarities or distances that are more discriminant.

Ordered Weighted Average

Another aggregation operator in this context is the ordered weighted average (Yager 1988). It associates weights to the respective positions of the dimension values instead of the dimensions themselves. This gives, in particular, more importance to the highest (or the lowest) values. This is important when aggregating matcher results, because this allows retaining only the results of the highest matches disregarding the dimension they come from.

Definition 7.13 (Ordered weighted average) Let o be a set of objects that can be analysed in n dimensions, an ordered weighted average operator f is a function from $D^n \rightarrow D$ (with D a set ordered by \leq and provided with an upper bound \top) satisfying $\forall x, x_1, \dots, x_n \in D$, defined as

$$f(x_1, \dots, x_n) = \sum_{i=1}^n w_i \times x'_i$$

such that

- w_1, \dots, w_n is a set of weights in $[0, 1]$ such that $\sum_{i=1}^n w_i = 1$;
- x'_i is the i th largest element of (x_1, \dots, x_n) .

The ordered weighted average has the properties of an average operator (commutative, monotone and idempotent). The max, min and average functions are special cases of ordered weighted average.

7.4.2 Voting

Aggregating alignments can be done by considering that each matcher is an independent source of information and that the decision to include a correspondence in an alignment is a vote in favour of this correspondence. This can be decided by a simple majority vote.

Definition 7.14 (Majority vote) Let $\{A_i\}_{i \in I}$ be a set of alignments over the same ontologies o and o' , the alignment A elected with majority from $\{A_i\}_{i \in I}$ is as follows:

$$A = \left\{ c \in \bigcup_{i \in I} A_i \mid |\{A_i \mid c \in A_i\}_{i \in I}| > \frac{|I|}{2} \right\}$$

This may be refined by taking into account the confidence associated to correspondences as weights.

Definition 7.15 (Majority weighted vote) Let $\{A_i\}_{i \in I}$ be a set of alignments over the same ontologies o and o' , the alignment A elected by majority weighted vote from $\{A_i\}_{i \in I}$ is as follows:

$$A = \left\{ c \in \bigcup_{i \in I} A_i \mid \sum_{i \in I} \kappa_i(c) > \frac{|I|}{2} \right\}$$

It is also possible to set a threshold with respect to the distributed weights, instead of majority.

In fact, any voting technique (Taylor 2005) may be applied to alignment aggregation and all previous summative measures considered before (weighted sum, weighted average, ordered weighted average) can be turned into a voting technique by adding a threshold.

Dempster-Shafer Theory

The Dempster-Shafer theory of evidence provides a numeric mechanism for modelling and reasoning with uncertain information. Its main strength is the proposition of a simple rule, known as the Dempster combination rule, which allows for a combination of evidences from independent sources (Dempster 1967; Shafer 1976).

In the Dempster-Shafer theory, a *sample space* is called a *frame of discernment* or just a *frame*. It is denoted by Ω and it consists of a set of hypotheses, which are to be unique and mutually exclusive. *Evidence* is used to choose the best hypothesis in Ω , or in other words, a piece of evidence that implies a hypothesis. Data sources, such as experts or sensors, provide such evidential assertions.

The evidence over a frame Ω is expressed through a *mass function* m , also known as a basic assignment function, namely, $m : 2^\Omega \rightarrow [0, 1]$, such that the following two conditions hold:

$$m(\emptyset) = 0,$$

$$\sum_{A \subseteq \Omega} m(A) = 1.$$

The mass is distributed in the $[0, 1]$ interval to the elements of the power set (2^Ω) of the propositions Ω . In turn, A is a subset of Ω and $m(A)$ represents the strength of an evidence supporting exactly the claim A . The first condition above means that the set Ω has to be complete; this corresponds to the closed-world assumption. The second condition means that expert statements have to be normalised, such that each source of evidence is equally important. Based on the mass function, several measures are defined.

In particular, beliefs in more specific propositions, such as subsets of A , are handled through a *belief measure*, $Bel : 2^{\mathcal{Q}} \rightarrow [0, 1]$, such that

$$Bel(A) = \sum_{B \subseteq A} m(B),$$

$Bel(A)$ is a measure of the total (thus, including its particular subsets) belief or justified support that is assigned to A . It captures all evidence B that supports the proposition A , i.e., that it is true.

In addition, there might exist another belief B , consistent (overlaps) with A . This is handled through a *plausibility measure*, $Pl : 2^{\mathcal{Q}} \rightarrow [0, 1]$, such that

$$Pl(A) = \sum_{A \cap B \neq \emptyset} m(B),$$

$Pl(A)$ is a measure of the maximum support that could be assigned to A , if justified with additional information.

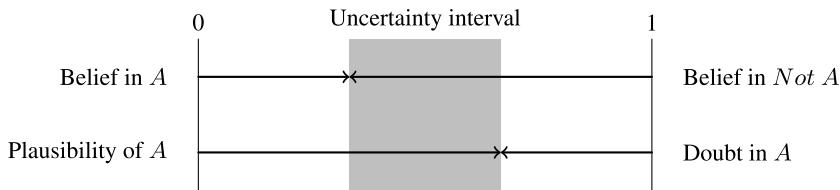


Fig. 7.9 Connections between belief, plausibility and uncertainty in the Dempster-Shafer theory.

The masses assigned by mass functions can be viewed as segments of the unit interval. The complete information about the measure of belief in the set A is expressed through the *belief interval* $[Bel(A), Pl(A)]$. In turn, $Pl(A) - Bel(A)$ represents *ignorance* (missing data) concerning A or the *uncertainty interval* (see Fig. 7.9).

The Dempster combination rule aggregates two mass functions by balancing combined evidences with conflicts:

Definition 7.16 (Dempster combination rule) Given two mass functions m and m' , the joint mass function $m \oplus m'$ of a non-null hypothesis A is defined as

$$m \oplus m'(A) = \frac{\sum_{B \cap B' = A} m(B) \times m'(B')}{1 - \sum_{B \cap B' = \emptyset} m(B) \times m'(B')}$$

Normalisation may lead to counter-intuitive conclusions when combining contradictory evidences (Zadeh 1984), since the rule emphasises the agreement between experts and neglects the conflicting evidence. A possible solution to this problem is to relax the closed world assumption (Smets 1990).

Example 7.17 (Usage of the Dempster-Shafer theory) The Dempster-Shafer theory has been used for combining various matching results in (Besana 2006; Wang et al. 2007; Nagy and Vargas-Vera 2010). In such a setting, similarity measures delivered by different matchers, such as edit distance and WordNet, are viewed as subjective expert assessments that provide the supporting evidences. Specifically, normalised similarity values correspond to mass values assigned to the pairs of entities to be matched and constitute elements of the frame. For example, $m_{wordnet}(\text{paper}, \text{article}) = 0.86$ means that according to a WordNet matcher, the mass function supporting the claim that paper corresponds to article results in the value of 0.86. The frame contains all possible correspondences assessed by the involved matchers. The masses provided by matchers are combined through the Dempster rule. Based on the combined evidences, the most likely correspondences from the mass distribution have to be selected. This can be done through thresholds with which, for instance, pairs of entities with very high plausibility and very low belief measures are rejected.

7.4.3 Arguing

Argumentation is a technique for finding an agreement between parties by exchanging arguments in favour of or against a particular position. In ontology matching, it may have two roles:

- negotiating an alignment between two agents, if they accept each other's arguments,
- achieving an alignment through matching. In particular, multiagent negotiation of alignments can be seen as another aggregation technique between two alignments. (Silva et al. 2005) presents such a system based on quantitative negotiation rather than arguments.

Argumentation allows agents to provide counter-arguments and to choose the arguments depending on their preferences. It has been used in ontology matching for finding an agreement between alignments (Trojahn et al. 2011). In this case, correspondences are seen as arguments and they tend to attack one another depending on whether they are contradicting each other and/or they are based on techniques preferred by the different agents. Hence, metadata on correspondences are important because they carry the basis for preferring or attacking a particular correspondence based on the provenance of the correspondence, the method with which it has been obtained, or the confidence which is attached to it.

Example 7.18 (Alignment argumentation) Consider two agents C and P using respectively ontology o and o' , expressed in description logic as follows:

$$o = \{\text{Micro-company} \equiv \text{Company} \sqcap \leq_5 \text{employee}\}$$

$$o' = \{\text{SME} \equiv \text{Firm} \sqcap \leq_{10} \text{associate}\}$$

Assume that they have discovered alignment A :

$$\begin{aligned} A = \{ & \langle \text{Company}, \text{Firm}, = \rangle, & (\gamma_1) \\ & \langle \text{employee}, \text{associate}, \leq \rangle, & (\gamma_2) \\ & \langle \text{Micro-company}, \text{SME}, \leq \rangle \} & (\gamma_3) \end{aligned}$$

The three correspondences are denoted, respectively, by γ_1 , γ_2 and γ_3 . The arguments in favour of γ_1 include

- a_1 : all the known Company on one side are Firm on the other side, and vice versa;
- a_2 : the two names Company and Firm are synonyms in WordNet.

The arguments in favour of γ_3 include

- a_3 : the alignment (without γ_3) plus the two ontologies entail the correspondence;
- a_4 : all the known micro-companies on one side are SME on the other side (and not vice versa);

and the counter-arguments include

- a_5 : the two names Micro-company and SME are not alike by any string distance, and they are not synonyms in WordNet;
- a_6 : the only features they share are associate and employee and they have different domains and cardinalities.

In (Laera et al. 2006), arguments are expressed following the value-based argumentation framework (Bench-Capon 2003). They are made of a flag denoting if they are in favour (+) or against (−) the correspondence and the type of method that supports this correspondence (basic methods). A simple way to express these arguments is as follows:

$$\begin{aligned} a_1: & \langle \text{Company}, \text{Firm}, =, & \langle +, \text{extensional} \rangle \rangle \\ a_2: & \langle \text{Company}, \text{Firm}, =, & \langle +, \text{terminological} \rangle \rangle \\ a_3: & \langle \text{Micro-company}, \text{SME}, \leq, & \langle +, \text{semantic} \rangle \rangle \\ a_4: & \langle \text{Micro-company}, \text{SME}, \leq, & \langle +, \text{extensional} \rangle \rangle \\ a_5: & \langle \text{Micro-company}, \text{SME}, \leq, & \langle -, \text{terminological} \rangle \rangle \\ a_6: & \langle \text{Micro-company}, \text{SME}, \leq, & \langle -, \text{structural} \rangle \rangle \end{aligned}$$

Such arguments could be delivered by existing basic matchers. Another, more elaborate way to define arguments is to allow correspondences themselves to be justifications. This is expressive enough to express, for instance, that the structural similarity of Micro-company and SME depends on the terminological similarity of employee and associate.

The rationale behind this type of argument is that some agents may prefer, or trust, better some techniques than others.

Example 7.18 (Alignment argumentation, continued) For instance, one can imagine that agent C prefers terminological arguments over extensional arguments, extensional arguments over semantic arguments and semantic arguments over structural arguments. This order induces a partial order on the arguments themselves: $a_5 \succ_C a_2, a_1 \succ_C a_2, a_5 \succ_C a_4, a_1 \succ_C a_4, a_2 \succ_C a_3, a_4 \succ_C a_3, a_3 \succ_C a_6$. Similarly, P could have a different preference ordering favouring structural, semantic, terminological and then extensional arguments.

There are logical theories (Dung 1995; Amgoud et al. 2000) that, given a set of arguments and the preferences of agents, define what is the consensus alignment between both parties. An argumentation framework is made of a set of arguments and an attack relation (Dung 1995). An argument a is called acceptable for a set of arguments S if any argument which attacks a is itself attacked by an element of S . A set of arguments is called admissible if none of its elements attacks another element and all its elements are acceptable. Finally, a *preferred extension* is an inclusion maximal admissible set.

The goal for agents is (i) to exchange arguments so that they can work on a common set of arguments, and (ii) to determine which arguments, and then positions, they can accept together. For that purpose, they may adopt a cautious approach, selecting only the arguments which are part of all their preferred extensions, or a credulous approach, selecting all arguments which are in at least one preferred extension of each agent.

Example 7.18 (Alignment argumentation, continued) For instance, C will have for preferred extension $\{a_5, a_1, a_2, a_6\}$ and P , in turn, will have $\{a_6, a_5, a_2, a_1\}$. However, together, the maximal common subset of arguments between C and P is $\{a_1, a_2, a_5, a_6\}$, which selects the preferred alignment made up of γ_1 and γ_2 .

Different results have been obtained using various refinements of argumentation: value-based argumentation, which allows for different preferences among arguments (Laera et al. 2006), strength value-based argumentation, vote value-based argumentation, which respectively builds preferences on confidences and votes across agents (Trojahn et al. 2008).

Summary on Similarity and Alignment Aggregation

The use of several matchers or similarity measures together is a common practice. In order to deliver an alignment, such measures have to be integrated. Integration may be based on aggregating similarity values or ranks. It can also be based on the opinion of stakeholders (matchers or agents), through simple voting or through more elaborate argumentation strategies.

Once integration has been performed, the resulting similarities can be used for extracting alignments (Sect. 7.7) or further composition and aggregation. They can also provide the basis for matcher learning (Sect. 7.5) and tuning (Sect. 7.6).

7.5 Matching Learning

This section is concerned with algorithms that learn how to sort alignments through the presentation of many correct alignments (positive examples) and incorrect alignments (negative examples). The main difference between these approaches is that the techniques of this section require some sample data to learn from. This can be provided by the algorithm itself and judged by users, for instance, by having only a subset of the correspondences under judgment, or this can be brought from external resources.

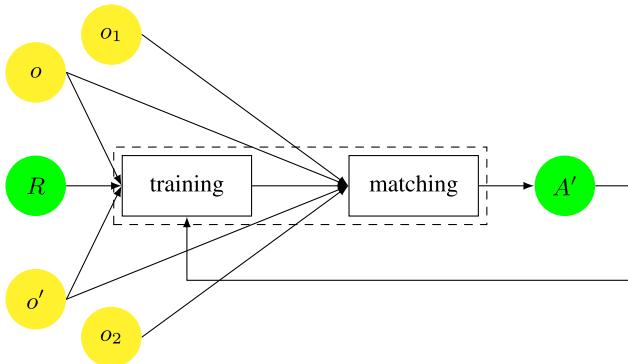


Fig. 7.10 General machine learning setting for ontology matching. In supervised machine learning, from a limited reference (R) of the alignment to obtain, the machine learning process generates an internal classification model, which is used for matching the remaining fragments of o and o' or the new ontologies o_1 and o_2 with similar characteristics to o and o' . In non-supervised machine learning, the resulting alignment (A') is usually injected as input to the training, which iterates until specific conditions are satisfied.

Matchers using machine learning usually operate in two phases (see Fig. 7.10): (i) the learning or training phase and (ii) the classification or matching phase. During the first phase, training data for the learning process is created, for example, by manually matching two ontologies, so that the system learns a matcher from this data. During the second phase, the learnt matcher is used for matching new ontologies. Feedback on the obtained alignment may be provided which can be fed into step (i) again. Learning can be processed on-line, such that the system can continuously learn, or off-line, if its speed is less relevant than its accuracy.

Usually this process is carried out by dividing a data set, i.e., set of positive and sometime negative examples of alignments into a training set (typically 80 % of data) and a control set (typically 20 % of data) which is used for evaluating the performances of the learning algorithm.

There are many types of information that a learner can exploit. These include word frequencies, formats, positions, or properties of value distributions. A multi-strategy learning approach is useful when several learners are used, each one han-

dling a particular kind of pattern that it learns best. Finally, results produced by various learners can be combined with the help of a metalearner (Doan et al. 2003).

In this section we consider some of the well-known machine learning methods which have been used for text categorisation, such as Bayes learning (Sect. 7.5.1), WHIRL learning (Sect. 7.5.2), neural networks (Sect. 7.5.3), support vector machines (Sect. 7.5.4), and decision trees (Sect. 7.5.5).

7.5.1 Bayes Learning

The naive Bayes learner (Good 1965) is a probabilistic induction algorithm. It has been used in various matching approaches as a classifier (Doan et al. 2004; Straccia and Troncy 2006; Lambrix and Tan 2006; Nandi and Bernstein 2009; Spiliopoulos et al. 2010; Esposito et al. 2010; Tournaire et al. 2011).

Assume that we want to match attribute x from one ontology to one (y_i) of the attributes (y_1, \dots, y_m , $i = 1, \dots, m$) from another ontology. The approach views values of attributes as sets of tokens. V denotes a set of underlying values of attribute x : $V = \{t_1, \dots, t_n\}$, where t_j is the j th token, $j = 1, \dots, n$. Tokens, in turn, are obtained by applying a normalisation technique, such as lemmatisation (Sect. 5.2.2), to the words in the data instance. Assume that $P(y_i)$ is the *a priori* probability that x matches y_i , i.e., without having seen any tokens of x . Then, $P(V)$ stands for the probability of observing values V in x . Finally, $P(V|y_i)$ stands for the conditional probability of observing values V , given that x matches y_i . The Bayes theorem describes how to optimally predict the attribute for a previously unseen data instance, given a training example. The chosen attribute is the one that maximises *a posteriori* probability, i.e., after having seen the values V , that x matches y_i . It is denoted by $P(y_i|V)$ and is computed as

$$P(y_i|V) = \frac{P(V|y_i) \times P(y_i)}{P(V)}$$

This is called the Bayes rule. The naive Bayes classifier has a *naive* assumption that the tokens t_j appear in V *independently* of each other given y_i . Based on this assumption the parameters (tokens) of each attribute can be learnt separately; this in turn greatly simplifies learning. Thus, if the attributes are independent given the class, $P(V|y_i)$ can be decomposed into the product of $P(t_1|y_i) \times \dots \times P(t_n|y_i)$ and $P(V)$ can be omitted from the Bayes rule for obvious reasons. Henceforth, the Bayes rule can be rewritten as follows:

$$P(y_i|V) = P(y_i) \times \prod_{1 \leq j \leq n} P(t_j|y_i)$$

The independence assumption often does not hold in practice. However, in many applications, the violation of this assumption does not lead to degradation in effectiveness of the approach (Domingos and Pazzani 1996).

The probabilities of the latter formula can be computed using the training data: $P(y_i)$ can be estimated by the proportion of examples that have been matched to y_i ; $P(t_j|y_i)$ can be estimated as $k(t_j, y_i)/k(y_i)$, where $k(y_i)$ is the total number of tokens of all training instances with attribute y_i , and $k(t_j, y_i)$ is the number of occurrences of token t_j in all training instances with attribute y_i . Based on the above formula the corresponding confidence scores can be designed in an obvious way.

Example 7.19 (Naive Bayes learning) Assume that we have established manually that attributes creator and name of one ontology match respectively attributes author and title of another one. The process works in two steps.

Training phase. Assume that Bertrand Russell is an instance of the creator attribute and My life is an instance of the name attribute. Thus, based on this information the following training examples can be fed into the classifier: $\langle\{\text{'Bertrand'}, \text{'Russell'}\}, \text{author}\rangle$ and $\langle\{\text{'My'}, \text{'life'}\}, \text{title}\rangle$. The second one declares that $\{\text{'My'}, \text{'life'}\}$ is a title and it has two tokens. By inspecting the training instances the learner builds its *internal classification model*. For example, by noticing that if a word such as ‘life’ occurs frequently in data instances positively related to title and infrequently in those related to other fields, their underlying attribute is therefore likely to match the title attribute on how to classify data instances. If the training set is statistically representative, these frequencies can be transformed into probabilities and the Bayes rule can be used. This can also be applied to classify instances in classes, for instance, using $\langle\{\text{title:My title:life}\}, \text{class:biography}\rangle$.

Matching phase. Let Life of Pi be an instance of the attribute h_1 from the structure of a web site which we want to match against attributes of the second ontology above. The learner uses its internal classification model to predict an attribute for the given instance as well as its confidence score, e.g., $\langle\text{author}, 0.2\rangle$, $\langle\text{title}, 0.8\rangle$. Based on the confidence scores, it can be concluded that h_1 is a match for title.

7.5.2 WHIRL Learner

WHIRL is an extension of conventional relational databases to perform *soft joins* based on the similarity of textual identifiers (not only based on equivalence of atomic values) (Cohen 1998). It has been also used for inductive classification of text and turns out to be competitive with other inductive classification systems, such as decision trees (Cohen and Hirsh 1998). The WHIRL approach to text classification can be viewed as a kind of nearest neighbour classification algorithm, such that all the training data is stored in memory and k instances are selected through the Euclidean distance and used for (k NN) classification (Nottelmann and Straccia 2006; Spiliopoulos et al. 2010).

WHIRL has been used in matching for learning both schema-level and instance-level information (Doan et al. 2003; Bilke and Naumann 2005). In the case of schema information, training examples could be of the following type (*expanded*

$\text{label}', \text{label} \rangle$, where label' belongs to ontology o' and label belongs to ontology o . For instance, $\langle \text{location}', \text{address} \rangle$ states that if an ontology entity has the label location, then it matches address. Expansion of label' can be obtained, for instance, by including its synonyms, which, in turn, can be obtained from manually created correspondence tables for the domain of interest. WHIRL stores all training examples it has seen. Assume that we would like to match another ontology o'' to ontology o . Given a label'' from o'' , WHIRL computes the corresponding label in o based on the labels of all examples in its collection that are similar to label'' . The similarity is based on TFIDF (Sect. 5.2.1) between the expanded labels of the examples. For example, given the label phone from o' , WHIRL may generate a prediction as follows: $\langle \text{address}, 0.1 \rangle$, $\langle \text{description}, 0.2 \rangle$, $\langle \text{agent-phone}, 0.7 \rangle$. Based on the confidence scores, it can be concluded that phone is a match for agent-phone.

In the case of instance-level information, this matcher uses the data content instead of expanded labels. A training example of this case is of the form $\langle \text{data instances}', \text{label} \rangle$, such that $\text{data instances}'$ belong to ontology o' and label belongs to ontology o . When matching a new ontology o'' to ontology o , the TFIDF distance between any two examples is the distance between data instances of o'' and the WHIRL collection of data instances.

7.5.3 Neural Networks

Artificial neural networks are made up of nodes (or neurons) and weighted connections between them. Nodes are grouped into layers, having input, output and either none, one or more hidden layers. Usually each node in a hidden layer is connected to all nodes of the preceding and the following layer. Neural networks have been widely used in practice due to their adaptability. Several types of neural networks have been used for various tasks in ontology matching, such as discovering correspondences among attributes via categorisation and classification (Li and Clifton 1994; Esposito et al. 2010) or learning matching parameters, such as matcher weights, to tune matching systems with respect to a particular matching task (Ehrig et al. 2005; Mao et al. 2010; Gracia et al. 2011). We focus here on the first task mentioned above, while learning matching parameters is addressed in Sect. 7.6.

Given schema-level and instance-level information, it is sometimes useful to cluster this input into m categories in order to lower the computational complexity of further manipulations with data. The *self-organising map* network and the corresponding self-organisation learning algorithm can be used for this purpose (Kohonen 2001). It categorises n nodes of the input layer into m categories of the output layer. Usually m is predefined based on how detailed the categories should be by setting the radius of clusters. Input patterns or attributes, e.g., field length and data type, are viewed as dimensions in a n -dimensional feature space. The neurons in the network are organising themselves according to the characteristics of given input patterns. This results in a clustered neuron structure, where neurons with similar properties are arranged in related areas on the map. Every node in the output layer represents a cluster centre.

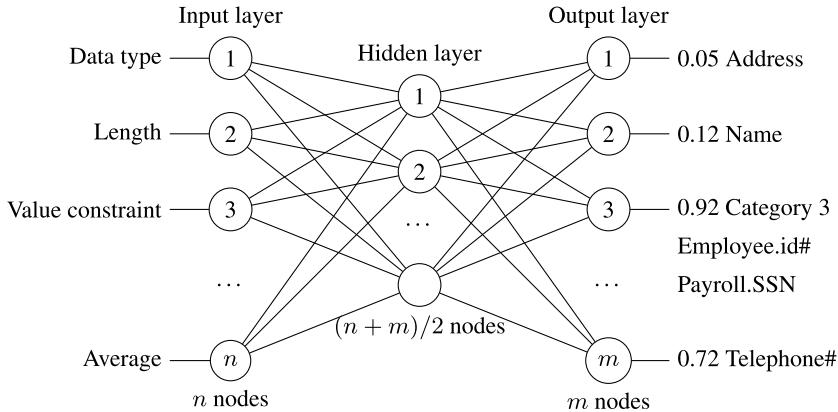


Fig. 7.11 Neural network identifying fields of `Health_Plan_Insured#` from their characteristics.

For neural networks, matching is viewed as a classification problem. The *back-propagation* algorithm can be used for this purpose. Back-propagation is a supervised learning algorithm which is used to train a network to recognise input patterns and give corresponding scores. First, the feature weights are loaded into the input nodes. Then, they are propagated forward in order to generate the output. If a misclassification occurs, the error is backpropagated in order to change the weights of connections in the network. Weights are modified until the errors in the output layer are not minimised anymore.

Example 7.20 (Neural networks—adapted from (Li and Clifton 1994)) Given an ontology, some of its attributes, such as `Employee.id`, `Dept.Employee` and `Payrol.SSN`, can be clustered into one category, since their input characteristics as well as intended meanings are close to each other. The corresponding vector of cluster centre weights can be as follows: $\langle 0, 0.1, 0, \dots \rangle$, where vector components stand for the features: the first position stands for data type, the second position stands for length, etc. The key feature for grouping the attributes mentioned above was the field length, since its value (0.1) is higher than that of others (0.0). In fact, ID fields are typically very regular (they use the full field all the time), while name fields are more variable.

Figure 7.11 shows a three-layer network for recognising m categories of patterns, given n features. The number of nodes in the hidden layer can be arbitrary. It is usually chosen based on experiments in order to obtain the shortest training time.

Training phase. The training data for the neural network is composed of vectors of cluster center weights and their target categories. For example, the vector considered previously, i.e., $\langle 0, 0.1, 0, 0, \dots \rangle$, is tagged with its target category, which is number 3. The back propagation algorithm will then adjust the weights so that attributes characteristics corresponding to these attributes will result in a output vector as close as possible to $\langle 0, 0, 1, 0, \dots \rangle$ indicating that the most likely category is 3.

Matching phase. The matching phase includes feeding into the network trained on features of ontology o a new pattern of n characteristics, e.g., of the attribute

`health_Plan.Insured#`, from another ontology o' . Based on the internal classification model of the network, it determines the similarity of this pattern and each of m categories. For instance, this attribute matches category 3 (id numbers) with the score 0.92 and category 1 with the score 0.05.

7.5.4 Support Vector Machines

Support vector machines (SVM) are supervised learning methods used for classification, regression and other tasks (Cristianini and Shawe-Taylor 2000). Basic SVMs are nonprobabilistic binary linear classifiers. However, they can efficiently perform nonlinear classification as well, by exploiting *kernel* functions. This implicitly transforms or maps SVM inputs into a higher-dimensional feature space, so that the originally linearly nonseparable data becomes (presumably) easier to separate in the feature space (Boser et al. 1992; Vapnik 2000). SVMs were used in several matching approaches as classifiers (Ehrig et al. 2005; Spiliopoulos et al. 2010; Tournaire et al. 2011).

Training instances are of two categories, those that match and those that do not match. Based on these, SVMs build a maximal separating hyperplane in a high-dimensional space, namely (w, b) , where w stands for the weight vector and b stands for bias. The decision function is $f(x) = w \times x + b$. The margin or the distance between the closest data points is maximised (which can be done also softly, namely by allowing for noise or erroneous examples (Cortes and Vapnik 1995)). The hyperplane decision function with kernel is

$$f(x) = w \times x + b = \sum_{i=1}^n y_i \alpha_i K(x_i, x) + b,$$

such that n is the number of training examples, $y_i \in \{+1, -1\}$ is the match or no-match label for the example i , K is a kernel function. α_i is a Lagrangian multiplier for each training point. $\alpha_i \geq 0$ and $\sum \alpha_i y_i = 0$. Only those training examples (x_i) that are positioned closely to the decision boundary have $\alpha_i > 0$, and those whose removal would change the solution found are called the *support vectors*.

Example 7.21 (Support vector machines) Given two ontologies, entity pairs to be matched are extracted, such as $\langle \text{name}, \text{title} \rangle$ or $\langle \text{id}, \text{isbn} \rangle$. For each of these, feature selection is performed through similarity calculation, e.g., by using SMOA, n -gram, or WordNet. The final result of this operation can be viewed as a similarity cube, where the first dimension stands for the entities of the first ontology, the second dimension stands for the entities of the second ontology and the third dimension stands for the features applied. For each entity pair, the target value is added, namely the match or no-match label. Both entities in the example above match. This is typically established by a domain expert, while preparing the training data set. In

other words, a vector is sliced from a similarity cube and give it a target value. These similarity vectors form a vector space, where training is performed to construct the maximal separating hyperplane. The selection of features or kernel type is crucial for the quality of the final result. Choosing a suitable kernel for a matching problem, i.e., a kernel that avoids irrelevant features, improves results. Various kernels can be used, such as polynomial, Gaussian radial basis functions or string and bag of words kernels. The matching phase includes feeding into the trained model new unseen features corresponding to entity pairs, which are classified by the SVM into match or no-match.

7.5.5 Decision Trees

Decision tree classifiers learn a set of rules which are applied in a sequential way and ultimately lead to a decision. Unlike the previous method, which are numeric in nature, and, therefore, not easily interpretable by humans, Decision trees do not have this drawback. A possible method for learning a decision tree for a category can follow a divide-and-conquer strategy. In a training set T of instances characterised by features and their category, a feature f_1 is selected, which discriminates the population in the best way (with regard to the set of categories). Then, T is partitioned into two subsets, the subset T_1^{yes} corresponding to feature f_1 , and the subset T_1^{no} without this feature. This procedure is recursively applied to T_1^{yes} and T_1^{no} . It stops if all instances in a subset are assigned to the same category. It generates a tree of rules with an assignment to actual categories in the leaves. Decision tree learner can be tolerant and accept that some of the instances are misclassified if this produces a large simplification of the tree. This is useful when there can be errors in the training sets.

Decision trees have been used in ontology matching for various tasks, such as discovering correspondences among entities (Xu and Embley 2003; Duchateau et al. 2009; Spiliopoulos et al. 2010; Tournaire et al. 2011) and learning parameters of matching systems, e.g., thresholds, to adapt automatically to a given matching task (Ehrig et al. 2005; Duchateau et al. 2008). We focus here on the former, while learning matching parameters is addressed in Sect. 7.6.

Example 7.22 (Decision tree) Given a large training alignment between instances from the two ontologies of Fig. 2.7, decision tree learning, e.g., the C4.5 decision tree induction system (Quinlan 1993), is applied to generate rules for identifying new instances. Figure 7.12 shows a decision tree fragment that can be learnt. The decision first states that it can only match Books from the first ontology into the second one. Then it distinguishes Books having one author, which is a Professor, from those having no Professor as authors. It is then able to consider that if an author is a topic of the Book, then this one must be classified as an Autobiography, otherwise it should be an Essay.

The decision tree has been built with some tolerance: the numbers after the target categories indicate the number of instances in the training set which have been correctly and incorrectly recognised.

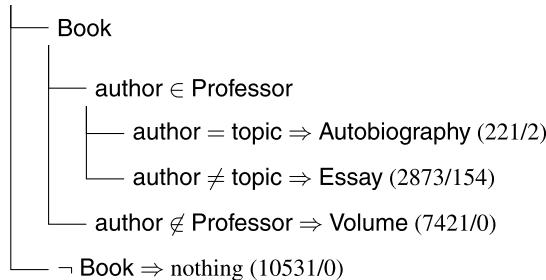


Fig. 7.12 Fragment of a binary decision tree. Each node is labelled by a condition that must be satisfied by the item to classify. When no further classification is possible, the resulting target category is indicated together with the number (in parenthesis) of correctly/incorrectly classified items in the training set.

The decision tree fragment displayed in Fig. 7.12 can be rewritten as mapping from the source ontology to the target ontology. The mapping rule corresponding to the Autobiography branch can be written as

$$\begin{aligned} \text{Book}(e) \wedge \exists e'; \text{author}(e, e') \wedge \text{Professor}(e') \wedge \exists e''; \text{author}(e, e'') \wedge \text{topic}(e, e'') \\ \Rightarrow \text{Autobiography}(e) \end{aligned}$$

Such a rule may be translated into a correspondence, albeit in an expressive language. It is possible to use the same kind of techniques for learning from the structure instead of instances. (Xu and Embley 2003) shows how to use decision trees in order to learn rules for matching terms in WordNet.

Summary on Matcher Learning

In this section we discussed basic matcher learning, which was essentially viewed as a classification problem. Many methods presented, such as Naive Bayes, k NN, SVM, and C4.5, have been used for matching by building on top of the Weka data mining software³ (Witten et al. 2011). Some specific frameworks, such as Neuroph⁴ for neural networks and libSVM⁵ for support vector machines were exploited

³<http://www.cs.waikato.ac.nz/ml/weka/>.

⁴<http://neuroph.sourceforge.net>.

⁵<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>.

as well with success for matching purposes. The discussed methods, their variations and other learning methods have been compared in-depth within the GLUE (Sect. 8.2.5), CSR (Sect. 8.3.21) and YAM++ (Sect. 8.3.23) matching approaches.

Technically speaking, setting up training and testing data sets, and then running directly a method from a package mentioned above, would rarely lead immediately to good results. Often, issues like overfitting and appropriate method parameter selection have to be addressed before achieving acceptable results. Application-wise, machine learning approaches are usually useful in data integration scenarios occurring in vertical domains (see Sect. 1.2), when new ontologies have to be matched against an existing pool of ontologies. Thus previously learnt matchers can be readily reused.

7.6 Matcher Tuning

Tuning refers to the process of adjusting a matcher for a better functioning in terms of:

- better *quality* of matching results, measured, for example, through precision, recall or F-measure (Sect. 9.3.1), and
- better *performance* of a matcher, measured through resource consumption, such as execution time, main memory, CPU or bandwidth (Sect. 9.3.4).

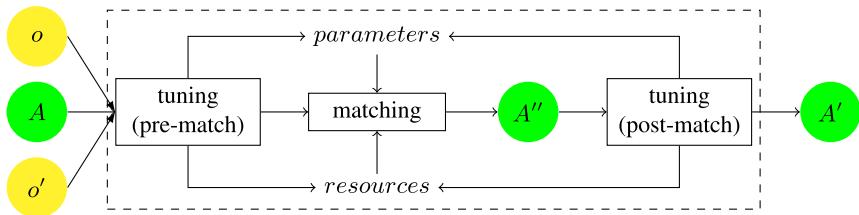


Fig. 7.13 General framework for tuning. Before matching, tuning decides in advance which matchers to employ, for example, based on the specific characteristics of the input ontologies. After matching, the output alignment (A'') of the matcher is evaluated by a tuner which may adapt the matcher parameters (and optionally resources). The matcher then produces a new alignment with the adapted parameters. This process iterates until the tuner cannot provide better parameters and returns the last computed alignment (A').

Tuning is usually conducted *before* matching, i.e., as a pre-match effort, *after* it, i.e., as a post-match effort, or *iteratively*, i.e., involving both or one of the two mentioned phases (see Fig. 7.13). This adjusting process can be automatic, semi-automatic or manual. Users perform tuning with the help of graphic interfaces or through direct editing of configuration files as a pre- or post-match efforts. For example, before matching, input ontologies may be analysed (manually or automatically) to obtain actionable insights, such as if they contain long or short labels, how

developed are their structures, if there are instances. This information is used to decide in advance dynamically (based on the input) to apply certain matchers or not, such as structure-level matchers, with respect to a default matching workflow. In turn, matcher selection is often tackled operationally by setting appropriate weights (in [0 1]) to matchers that are in a predefined pool (of usually at most several dozens of matchers) and to be further aggregated. So far, mostly design-time toolboxes offer manual weight selection (Do and Rahm 2007; Gal and Shvaiko 2009). Tuning is often automated through *predefined rules* (Mochol and Jentsch 2008; Li et al. 2009; Peukert et al. 2012) or *machine learning* (Lee et al. 2007; Duchateau et al. 2008; Gal 2011). Implementations might tightly couple tuning and matching, such that it may appear as a run-time activity, while conceptually tuning is positioned before or after matching.

From a methodological point of view, tuning may be applied at various levels of architectural granularity. For example, for choosing a specific matcher, such as edit distance, from a library of matchers, for setting parameters of the matcher chosen, e.g., cost of edit distance operations (Sect. 5.2.1), for aggregating the results of several matchers, e.g., through weighting (Sect. 7.4.1), for enforcing constraints, such as 1:1 alignments (Sect. 7.4.1), or for selecting the final alignment, e.g., through thresholds (Sect. 7.7.1). In all these examples, informed decisions, for instance, for choosing a specific threshold of 0.55 vs. 0.57 vs. 0.6, should be made. When a library of matchers is offered by a tool, it may be challenging for users to select, compose and parameterise them (Sect. 3.4), because the solution space is too large to try out all the alternatives. Hence efforts have been made for achieving this automatically. This is sometimes called ontology metamatching (Lee et al. 2007; Eckert et al. 2009). We consider the above mentioned operations together because, in practice, they always operate among a predefined number of choices which can often be reduced to a (sophisticated) variant of parameter tuning.

As the definition of the matching process indicates (Sect. 2.5.1), a matcher, besides two input ontologies, may also use an input alignment A , parameters, p , e.g., weights, and external resources r . These three elements constitute a matcher *configuration*, $cfg = \{(A, p, r)\}$, which can be tuned. For simplicity its three elements can be generalised and represented by a set of pairs:

$$cfg = \{\langle p_1, v_1 \rangle, \dots, \langle p_n, v_n \rangle\},$$

of parameters (p_i) and their values (v_i), such as a threshold of 0.55. In turn, \mathcal{CFG} is a space of all possible configurations. The goal of tuning is to discover an optimal configuration, namely with which changing parameters does not decrease matching quality, e.g., precision, or matcher performance, e.g., execution time, or both. For example, optimising matching quality through F-measure ($M_{.5}$; Sect. 9.3.1) means searching for a cfg_* in the \mathcal{CFG} space, that improves F-measure:

$$cfg_*^{i+1} \in \{cfg \in \mathcal{CFG} : M_{.5}(cfg) \geq M_{.5}(cfg_*^i)\}$$

Usage Examples eTuner (Sect. 8.4.4) uses an approach to tune a library of schema matchers at design time: given a particular matching task, it automatically tunes a

matching system by choosing suitable matchers, and the best parameters to be used, such as thresholds. MatchPlanner (Sect. 8.1.39) uses a decision tree as an aggregation function, such that the nodes represent similarity measures and edges are used as conditions on the results. Such a decision tree represents a plan whose elementary operations are matching algorithms. Moreover, since edges in the decision tree are used as conditions, these can be viewed as thresholds, personalised to each matcher. In turn, ECOMatch (Ritze and Paulheim 2011) uses an alignment sample provided by users and optimises partial F-measure with respect to this sample. It considers matchers as black boxes and uses out-of-the-box optimisation techniques to find optimal parameters of these systems. This allows for selecting both an optimal matcher and parameters. AMS (Sect. 8.4.10) implements a rule-based approach to adapt a matching process: first the matching workflow is established with respect to the characteristics of the input ontologies, then the output is evaluated in order to adapt the workflow. Characteristics of both the input and the output are considered features accessible to rules. Rules are triggered in a particular context and modify the workflow, e.g., by adding aggregators (when there are several results), suppressing basic matchers (when their quality is not sufficient).

Below we discuss in some detail and with examples two specific methods to perform tuning through aggregation of basic matchers. These are stacked generalisation (Sect. 7.6.1) and genetic algorithms (Sect. 7.6.2).

7.6.1 Stacked Generalisation

Stacked generalisation is an approach to combine multiple learning algorithms (Wolpert 1992). From the ontology matching perspective, this approach can learn to aggregate several basic learners, e.g., naive Bayes and WHIRL, on a particular label (Doan et al. 2003; Esposito et al. 2010).

The training phase of stacked generalisation works in two steps. The first step deals with collecting the output of each learner, thereby resulting in a new data set. First, let $D^0 = \{(c_i, x_i)\}_{i=1,\dots,m}$ be the training data set, such that c_i is a category and x_i is an instance represented by the vector of its features. In terms of ontology matching, c_i can be an entity from ontology o and x_i an entity of ontology o' , e.g., an individual represented by its attributes. c_i is the category to which x_i should be assigned or, for ontology matching, the entity corresponding to x_i . In order to avoid overfitting, i.e., that the training data does include the query instances, a *cross-validation* is performed. In particular, D^0 is randomly partitioned into p almost equal parts, D_1^0, \dots, D_p^0 , such that D_k^0 represents a test and $\bar{D}_k^0 = D^0 - D_k^0$, represents a training set for the k -th class of cross-validation. Given q basic learning algorithms (matchers), which are called *level-0 generalisers*, using the l -th matching algorithm on training set \bar{D}_k^0 results in a *model* M_l^k , which has been learnt. Such a model, given a vector of features characterising an object, returns a prediction which is the category it should be assigned to. These are the *level-0 models*. Let M_l^i be the prediction of the model M_l^k on $x_i \in D_k$. The final result of cross-validation is the set D^1 which consists of exactly one prediction for each of the training examples:

$$D^1 = \{\langle c_i, \langle M_1^i, \dots, M_q^i \rangle \rangle\}_{i=1,\dots,m}$$

The output of the first step is used as the input data for the second step, where another learning algorithm, called *level-1 generaliser*, is employed. In turn, it derives a *level-1 model* M' for c_i with respect to M_1^i, \dots, M_q^i . Thus, while level-0 classifiers deal with the possible assignment of entities to categories with regard to their attributes, level-1 classifiers deal with the possible assignment of the same entities to the same categories with regard to the categories predicted by the classifiers.

During the classification phase, given a new instance x , the models which have been learnt produce a vector $\langle M_1, \dots, M_q \rangle$, which is taken as input by M' , whose output is the final classification for that instance.

(Ting and Witten 1999) identified that the best results, compared to such learning algorithms as C4.5 decision tree (Sect. 7.5.5) or naive Bayes (Sect. 7.5.1), are obtained in stacked generalisation for classification tasks when (i) the higher-level model combines the *confidence* and not the predictions of the lower-level models, and (ii) the *multiresponse linear regression* is used as a level-1 generaliser.

Besides multiresponse linear regression, there are other algorithms that may work equally well on this task, such as neural networks (Sect. 7.5.3).

Example 7.23 (Stacked generalisation, adapted from (Doan et al. 2003)) Assume that two basic learners are used: (i) the WHIRL learner working with labels of entities (Sect. 7.5.2), and (ii) the naive Bayes learner working with data instances of entities (Sect. 7.5.1). The names of these matchers are abbreviated in this example as WHIRL and NB respectively.

Training phase. Consider the label address from ontology o . Examples of corresponding training data from ontology o' for basic learners are shown in Table 7.1. The first and the second columns list respectively the labels, e.g., location, and the underlying data instances, e.g., $\langle \text{Miami}, \text{FL} \rangle$, of some entities from ontology o' . The fourth and the fifth columns describe confidence scores S as produced by WHIRL and naive Bayes based on input from the first three columns via the cross-validation. For example, $S_{\text{WHIRL}}^{\text{address}}(\text{location}) = 0.5$, while $S_{\text{NB}}^{\text{address}}(\langle \text{Miami}, \text{FL} \rangle) = 0.8$. Finally, the last column indicates whether the correspondence under consideration holds or not. For example, location from o' actually matches address from o , and therefore, the corresponding value in the last column is 1, while listed-price does not match address, and therefore, the corresponding value in the last column is 0.

The information from the three right-most columns is used as input for the linear regression (Breiman 1996; Birkes and Dodge 2001). Results of the WHIRL and naive Bayes learners stand for the confidence scores (S), while the last column represents values of the response variable. As the result of least square error minimisation the weight assigned to the pair of WHIRL and label address is 0.2, i.e., $W_{\text{WHIRL}}^{\text{address}} = 0.2$, while $W_{\text{NB}}^{\text{address}} = 0.9$. The interpretation of these weights is that, based on stacked generalisation training, the naive Bayes learner appears to be much more reliable compared to WHIRL in its predictions about address.

Table 7.1 Training data for basic learners and stacked generalisation.

o' label	o' instance	o label	WHIRL	NB	True predictions
location	Miami, FL	address	0.5	0.8	1
listed-price	250K	address	0.4	0.3	0
phone	(305) 729 0831	address	0.3	0.6	0
comments	Fantastic house	address	0.6	0.1	0
location	Boston, MA	address	0.5	0.9	1
listed-price	320K	address	0.2	0.2	0

Matching phase. Assume that we want to match the entity with label area and instance $\langle \text{Seattle, WA} \rangle$ from yet another ontology o'' to entities of ontology o . Consider the case of the entity with label address from o . WHIRL will analyse the label area and generate its confidence score, e.g., $\langle \text{address}, 0.4 \rangle$. The naive Bayes learner, in turn, will analyse the data contents and generate its confidence score, e.g., $\langle \text{address}, 0.8 \rangle$. By using the weights obtained during the training phase of stacked generalisation, the weighted average of the confidence scores can be computed as follows: $0.4 \times 0.2 + 0.8 \times 0.9 = 0.8$, yielding the combined prediction $\langle \text{address}, 0.8 \rangle$.

There are other techniques with similar goals, such as *boosting* and *bagging*. To combine the decisions of the individual models (matchers), boosting uses a weighted majority vote and bagging uses unweighted majority vote. However, they require a large number of models because they rely on varying the data distribution to obtain a diverse set of models from a single learning algorithm, while stacking can work with only a few level-0 models (Ting and Witten 1999). The application of boosting (through the AdaBoost algorithm) in order to select matchers from a pool to be further used in combination was investigated in the SMB system (Sect. 8.4.8).

7.6.2 Genetic Algorithms

Genetic algorithms are population-based computational models inspired by genetics and evolution (Holland 1992). These are adaptive heuristic search methods often used for complex parameter optimisation problems. They essentially perform a randomised global search in a solution space (Mitchell 1996).

Individuals of a population are viewed as solutions and the environment they live in corresponds to the problem objectives and constraints to which these individuals are being adapted. In the selection process the *survival of the fittest* principle is followed. Through generations of populations, which are called iterations, preferable attributes of individuals (promising solutions) are pushed ahead in evolution, while weaker solutions are driven to extinction from the population. The key building blocks of genetic algorithms are as follows:

Encoding. The original problem space is known as the *phenotype* space, which is to be encoded to its genetic space counterpart, known as the *genotype* space, where the search is performed. Before delivering its result, genotypes should be decoded to the phenotype space. Individuals are represented as genotype-like data structures in a population as a binary, integer or real-valued string. Each string element stands for a particular feature of the solution.

Population initialisation. The first population is initialised by a random generation of, for instance, 1000 individuals, though some problem specific heuristics can be used as well, thereby improving an initial individual fitness.

Evaluation. Each individual in a population is evaluated through a fitness function in order to determine its quality and to decide the respective reproductive opportunities, namely what critical information to preserve and pass to the offspring. This results in decreasing ordering of 1000 individuals based on the values of the fitness function.

Parent selection. Parents are selected to produce offspring. For example, this can be done randomly by choosing individuals with a probability being directly proportional to their fitness, which is known as a roulette wheel sampling. For instance, out of 1000 ordered individuals, the first half can be selected as parents.

Crossover. This is a variation operation used to combine the features of the two selected parent genotypes to create one or two offspring genotypes. This is (often) a stochastic operation.

Mutation. This is another variation operation used to alter one parent genotype to create a slightly modified offspring of it. This is a stochastic operation. It is motivated by the possibility of reinserting the lost (in generations) information in order to prevent premature convergence without reaching a satisfactory solution and to foster genetic diversity.

Population generation. New individuals are created through crossover and mutation to build the next generation. Individuals of the population is then evaluated for its fitness (see *Evaluation* step).

Termination condition. A suitable termination condition should be determined, e.g., by reaching the desired precision of the fitness level or once a fixed number of iterations yielding no fitness improvements. This is typically handled through thresholds.

Genetic algorithms were used for different purposes in the context of ontology matching. For example, they were used in (Vázquez-Naya et al. 2010) to search for the optimal combination of multiple matchers in order to aggregate the results of these into a single value, or in (Wang et al. 2006) and (Elmeleegy et al. 2008), to determine approximately the globally optimal alignment with respect to similarity features which constitute the fitness function, such as extensional or usage-based similarity. Finally, genetic algorithms have been used as a near-optimal alignment extraction strategy (Sect. 7.7) for alignment selection (Qazvinian et al. 2008).

Example 7.24 (Genetic algorithms) Consider the use of genetic algorithms for optimal aggregation of multiple matchers (Vázquez-Naya et al. 2010). The task is to

discover the weights that maximise the global alignment quality. Each individual represents a potential solution, namely a set of weights, the sum of which should be equal to 1. The encoding of the problem is as follows: each position in the genotype contains a value in the [0 1] range, which represent a *separation* point that bounds a matcher weight. The decoding of a genotype is performed in ascending order of the separation values. The process starts with randomly generated aggregation, namely a set of the respective weights. For instance, we have to aggregate the results of five different matchers and the i th genotype composition can be of the following four separation points: 0.51, 0.11, 0.92, 0.57 whose decoding (by re-ordering) is as follows: 0.11, 0.51, 0.57, 0.92. The matcher weights are computed as differences between the separation points, resulting in: $w_1 = 0.11$, $w_2 = 0.40$, $w_3 = 0.06$, $w_4 = 0.35$, $w_5 = 0.08$. F-measure is used as a fitness function. Parent selection is made through the roulette wheel method. Crossover passes to the next generation only those descendants whose fitness exceeds that of the parents. Mutation is done with low probabilities, by substituting a genotype element with a randomly generated one. The best individuals from one generation are copied to the next one. The process stops when the individual fitness function is higher than a threshold. The fittest individual is ultimately decoded as the selected solution.

Summary on Matcher Tuning

In this section we introduced a general framework for tuning and we discussed two specific methods, such as stacked generalisation and genetic algorithms, used to adjust matching systems for a better functioning through aggregation of basic matchers. This topic has emerged in the recent years and deserves further investigations. In fact, often, there are many different constraints and requirements applied to the matching tasks, e.g., correctness, completeness, execution time, and main memory, thereby involving multi-criteria decision. The main issue is the semi-automatic combination of matchers by looking for complementarities and balancing the weaknesses and reinforcing the strengths of the components. For example, the aggregation is usually performed following a pre-defined aggregation function, such as a weighted average. The work on evaluation (Sect. 9) can be used in order to assess the strengths and the weaknesses of individual matchers by comparing their results with task requirements. Novel ways of performing aggregation with provable qualities of alignments have to be looked for.

7.7 Alignment Extraction

The goal of matching is to identify a satisfactory set of correspondences between ontologies. A (dis)similarity measure between the entities of both ontologies provides a large set of correspondences. Those which will be part of the resulting alignment remain to be extracted on the basis of the similarity. This can be achieved

by a specialised extraction method which acts on the similarity matrix or on some pre-alignment already extracted. We distinguish between the extractor itself, which converts a (dis)similarity matrix into an alignment, and a filter, which reduces the candidate correspondences in one of these formats. This is depicted in Fig. 7.14.

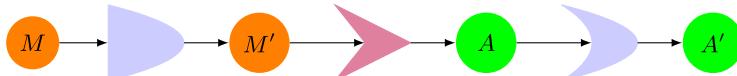


Fig. 7.14 Similarity filter, alignment extractor and alignment filter.

Similarity filters transform the (dis)similarity matrix by, for instance, by zero-ing all cells under some threshold or by unit-ing those above a threshold. Alignment extractors generate an alignment from a similarity matrix. They are the main topic of this section. Alignment filters can further manipulate alignments by using the same types of operations as similarity filters.

Users can act as alignment filters: an alignment may be obtained by displaying the entity pairs with their similarity scores and ranks, leaving the choice of the appropriate pairs up to users. This user input can be taken as the definitive answer in helper environments, as the definition of an anchor for helping the system (Sect. 8.1.9) or as relevance feedback in learning algorithms (Sect. 7.5).

One could go a step further and attempt to define algorithms that automate alignment extraction from similarity scores. Various strategies may be applied to the task depending on the properties of the target alignment.

This problem can be defined as follows:

Definition 7.25 (Alignment extraction problem) Given two sets of entities o and o' and a similarity function $\sigma : o \times o' \rightarrow [0, 1]$, extract an alignment $A \subseteq o \times o'$.

This problem statement is underconstrained since $o \times o'$ is a solution to this problem. So this section considers how to further constrain the problem of alignment extraction. One guide for doing so has been introduced in Sect. 2.5.2 as the totality and injectivity constraints on alignments.

We present two main strategies based on trimming the (dis)similarity after some threshold (Sect. 7.7.1) and on determining an optimal overall (dis)similarity of the extracted alignment (Sect. 7.7.3). In between, we present a kind of filter that has been found useful in matching algorithms (Sect. 7.7.2).

7.7.1 Thresholds

If neither ontology needs to be completely covered by the alignment, a threshold-based filtering would retain only the most similar entity pairs. Without the injectivity constraint, the pairs scoring above the threshold represent a sensible alignment.

Thus, applying thresholds requires that the extracted alignment is of sufficient quality. An easier way to proceed consists of selecting correspondences over a particular threshold. Several methods can be found in the literature (Do and Rahm 2002; Ehrig and Sure 2004):

- Hard threshold** retains all correspondences above threshold n ;
- Delta threshold** consists of using as a threshold the highest similarity value out of which a particular constant value n is subtracted;
- Gap threshold** retains the correspondences ordered by decreasing similarity until the difference in similarity between two correspondences becomes larger than n ;
- Proportional threshold** consists of using as a threshold the percentage of the highest similarity value;
- Percentage** retains the $n\%$ correspondences above the others.

The Rondo system (Sect. 8.1.14) provides an original alignment extraction method (`SelectThreshold`) which normalises the similarity of each node by the best similarity it has with another node (the result is not symmetric anymore). It then selects for the alignment the pairs of nodes for which the normalised similarity of both nodes is above some defined threshold.

Example 7.26 (Thresholding methods) We start from the weighted sum distance obtained in Example 7.9 with $1/4$ – $3/4$ weights. This distance is converted into a similarity as in the following table:

	Book	Translator	Publisher	Writer
Product	.19	.05	.03	.04
Provider	.04	.05	.75	.14
Creator	.05	.25	.04	.69

- A hard threshold of .23 would select the $\langle \text{Provider}, \text{Publisher}, = \rangle$, $\langle \text{Creator}, \text{Writer}, = \rangle$, and $\langle \text{Creator}, \text{Translator}, = \rangle$ as correspondences.
- A delta threshold with the same .23 value would select only the first two and the corresponding hard threshold would be $.75 - .23 = .52$.
- A gap threshold of .23 would select the same two correspondences because $.69 - .25 > .23$.
- On the contrary, the use of a proportional threshold of .23 would result in a $.75 \times .23 = .17$ hard threshold, so selecting $\langle \text{Product}, \text{Book}, = \rangle$ in addition to the three ones above.
- The percentage threshold of .23 would select the $12 \times 23 \% \approx 3$ initially selected pairs.
- The `SelectThreshold` method for a threshold of .23 would also yield the set of four correspondences mentioned above.

7.7.2 Strengthening and Weakening

Some approaches, such as (Ehrig and Sure 2004), use a sigmoid function between 0. and 1. ($\text{sig}_a(x) = 1/(1 + e^{-a(x-0.5)})$) with a being a parameter for the slope). This allows reinforcing values higher than 0.5 and weakening those lower than 0.5. This treatment is meant to clearly separate two zones: the positive and negative correspondences (see Fig. 7.15).

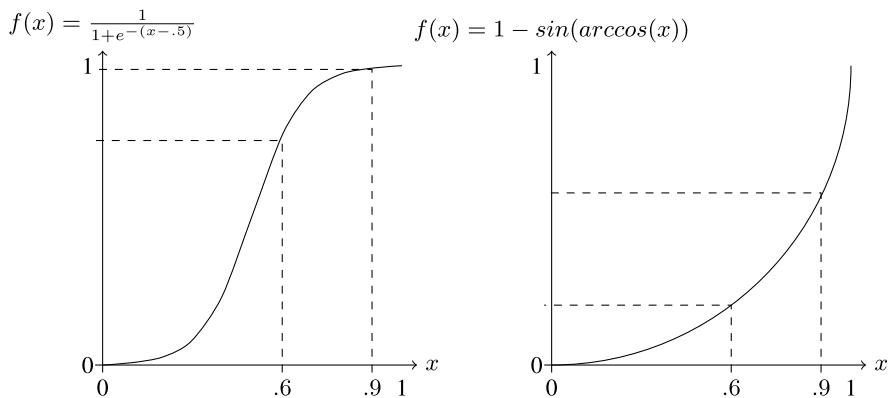


Fig. 7.15 Sigmoid and trigonometric smoothing functions.

Other functions, such as $1 - \sin(\arccos(x))$, can have an opposite effect: discarding the nonconclusive measures and dispatching the highest ones on the unit interval. This treatment is well justified by considering that very similar entities are indeed similar but loosely similar entities give nonconclusive results. Of course, it is possible to shift these functions in order to select threshold other than .5.

Example 7.27 (Strengthening and weakening) The two tables below display the similarity table of Example 7.26 filtered through the two functions displayed in Fig. 7.15.

	Book	Translator	Publisher	Writer		Book	Translator	Publisher	Writer
Product	.02	0.	0.	0.	Product	.02	0.	0.	0.
Provider	0.	0.	.95	.01	Provider	0.	0.	.34	.01
Creator	0.	.05	0.	.91	Creator	0.	.03	0.	.28
Filtered by $y = \frac{1}{1+e^{-12(x-.5)}}$					Filtered by $y = 1 - \sin(\arccos(x))$				

The sigmoid function provides high values for the best matches and lower ones for the worse matches while the other proposed function requires higher similarities than .75 to single them out: it reduces all values.

7.7.3 Optimising the Result

If an injective alignment is required then some choices need to be made in order to maximise the quality of the alignment. This quality is typically measured on the total similarity of the matched entity pairs. Consequently, the matching algorithm must optimise the global criteria rather than maximising the local similarity at each entity pair.

To sum up, the alignment computation can be viewed as a less constrained version of the basic set similarity functions *MSim* (Sect. 5.4.3). Indeed, its target features are the same: (i) maximal global similarity, (ii) exclusivity, and (iii) maximal cardinality (in correspondences). However, (ii) and (iii) are not mandatory: they depend on injectivity and totality requirements, respectively. Extracting an alignment from a similarity table is typically what is called graph matching (Berge 1970; Lovász and Plummer 1986) and more precisely weighted bipartite graph matching (for injective alignments) or covering (for total alignments).

A greedy alignment extraction algorithm could construct the correspondences stepwise, at each step selecting the most similar pair and deleting its members from the table. The algorithm will then stop whenever no pair remains whose similarity is above the threshold. The greedy strategy is not optimal (see Example 7.31); however the ground on which a high similarity is forgotten to the advantage of lower similarities can be questioned and thus the greedy algorithm could be preferred in some situations.

There are two notions of optimal matching of two sets in this context: the first one is a local optimum called *stable marriage* and the second one is a global optimum called *maximum weight matching*.

A stable marriage is an assignment (of only one object of the first set to only one object of the second set), such that there is no pair of entities, involved in different correspondences, which both may prefer to be put in a correspondence, i.e., the similarity between these two entities is higher than the similarity each of them has with the entity they are in correspondence. Algorithms for computing stable marriages are the Gale–Shapley algorithm (Gale and Shapley 1962) and the greedy algorithm presented above.

Definition 7.28 (Stable marriage problem) Given two sets of entities o and o' and a similarity function $\sigma : o \times o' \rightarrow [0, 1]$, extract a one-to-one alignment from $M \subseteq o \times o'$ such that for any $\langle p, q \rangle \in M$ and $\langle r, s \rangle \in M$, $\sigma(p, q) \geq \sigma(p, s)$ or $\sigma(r, s) \geq \sigma(r, q)$.

The greedy algorithm guarantees that the found solution is a stable marriage. However, we introduced a slightly different problem, in which the constraint is that any permutation of two assignments provides a worse result.

Definition 7.29 (Pairwise maximal matching problem) Given two sets of entities o and o' and a similarity function $\sigma : o \times o' \rightarrow [0, 1]$, extract a one-to-one alignment from $M \subseteq o \times o'$, such that for any $\langle p, q \rangle \in M$ and $\langle r, s \rangle \in M$, $\sigma(p, q) + \sigma(r, s) \geq \sigma(p, s) + \sigma(r, q)$.

A pairwise maximal assignment is not necessarily a stable marriage and vice versa (see Example 7.31).

The second notion is the global optimum, or *maximum weight matching*. It is an assignment for which there does not exist any other assignment with better weighting. It can be computed by the Hungarian method (Munkres 1957).

Definition 7.30 (Maximum weight graph matching problem) Given two sets of entities o and o' and a similarity function $\sigma : o \times o' \rightarrow [0, 1]$, extract a one-to-one alignment from $M \subseteq o \times o'$, such that for any one-to-one alignment $M' \subseteq o \times o'$,

$$\sum_{\langle p, q \rangle \in M} \sigma(p, q) \geq \sum_{\langle p, q \rangle \in M'} \sigma(p, q)$$

If weights represent dissimilarities instead of similarities, the problem to solve is the dual minimum weight graph matching.

Example 7.31 (Stable marriage, pairwise maximal and maximum weight matchings) Consider the following similarity table for the concepts of Fig. 6.3 from which we want to extract a one-to-one alignment.

	Book	Translator	Publisher	Writer
Product	.84	0.	.90	.12
Provider	.12	0.	.84	.60
Creator	.60	.05	.12	.84

The greedy algorithm would select first the highest scoring (.90) correspondence $\langle \text{Product}, \text{Publisher} \rangle$ and discard the corresponding line and column. It would then select the next highest scoring (.84) one, $\langle \text{Creator}, \text{Writer} \rangle$, and then the remaining best one, $\langle \text{Provider}, \text{Book} \rangle$. The result is clearly a stable marriage (checking that every pair selected cannot be broken by symmetric individual preferences is left as an exercise) but not a pairwise maximal matching. The alignment made of these three correspondences scores 1.96.

However, there are better alignments. For instance, by replacing the last two elements with $\langle \text{Creator}, \text{Book} \rangle$ and $\langle \text{Provider}, \text{Writer} \rangle$, we obtain an alignment scoring 2.1. This alignment is pairwise maximal, but not a stable marriage.

This is not, however, the maximum weight matching, which is made of $\langle \text{Product}, \text{Book} \rangle$, $\langle \text{Provider}, \text{Publisher} \rangle$, and $\langle \text{Creator}, \text{Writer} \rangle$ and scores 2.52. This assignment is pairwise maximal, but not a stable marriage.

Summary on Alignment Extraction

Alignment extraction is usually a necessary step for similarity-based matching: a similarity is not an alignment. The techniques which were presented apply to any similarity or dissimilarity measure. They range from simple, e.g., those applying thresholds, to more sophisticated methods, such as solving optimisation problems. There is no absolute criterion for choosing one of these methods: their relevance often depends on the application. An important criterion is the form of the expected final result.

7.8 Alignment Improvement

Alignment improvement consists of measuring some quality of a produced alignment, reducing the alignment, so that the quality may improve, and possibly iterating by expanding the resulting alignment. A general improvement process is presented in Fig. 7.16. Because there may be several possible reductions of the initial alignment, the one to be used has to be selected. When the measure reaches a particular threshold, the alignment is selected as the result of this process. If the selected alignment is directly provided as the output instead of being fed back to the matcher, then improvement acts as an alignment filter (Sect. 7.7).

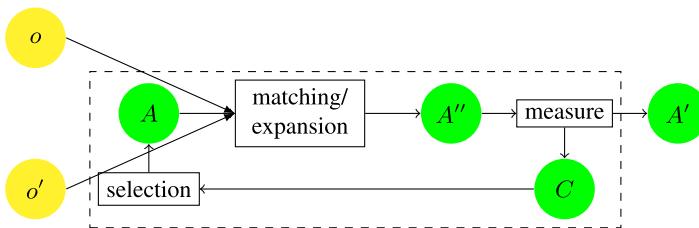


Fig. 7.16 General alignment improvement framework. An initial alignment (A'') is produced by a matcher. It is evaluated with respect to an intrinsic measure (consistency, agreement, constraint violation), which determines sets of compatible subalignments (C). Among these, one is selected and fed back in the process as input alignment (A), so that the matcher can improve it. This process is iterated until the measure does show a satisfying value and the last computed alignment (A') is returned as final.

The improvement process can be carried out manually with users controlling each of the steps and possibly interleaving other operations or automatically with

matchers embedding improvements. Users may contribute (i) by indicating that a particular set of correspondences is not correct (in replacement of the measure) or (ii) by taking the place of the selection component.

Quality measures are the main ingredients for improvement. Contrary to other measures described in Sect. 9.3, these must be *intrinsic* measures of the alignment: they do not depend on any reference. Various such measures may be used:

- threshold on confidence or average confidence,
- cohesion measures between matched entities, i.e., their neighbours are matched with each other,
- ambiguity degree, i.e., proportion of classes matched to several other classes,
- agreement or nondisagreement between the aligned ontologies (d'Aquin 2009),
- violation of some constraints, e.g., acyclicity in the correspondence paths,
- satisfaction of syntactic antipatterns (see Sect. 6.1.4),
- consistency and coherence (Sect. 2.5.3).

7.8.1 Alignment Disambiguation

An example of such a method is alignment disambiguation. An alignment is ambiguous, when some entities are matched with several other entities (assuming the relation is equivalence), e.g., a $??:?$ alignment is sought but a $*:*$ alignment has been returned.

So the measure may be an ambiguity degree, i.e., the proportion of classes matched to several other classes, or a Boolean indicator, i.e., the alignment is ambiguous or not. A simple method for dealing with this problem is to always choose the correspondence with the higher confidence (greedy algorithm) or to compute a maximal weight subalignment (Sect. 7.7.3).

An alternative solution is based on the idea that the correct match among two classes is prone to having other correct matches among its more general and more specific entities (Tordai 2012). So, for each of the ambiguous correspondences, the method counts the proportion of correspondences reachable by subclasses of the related classes, that are reachable from both sides. Then, it keeps the correspondence that has the highest ratio. Hence for any ambiguous pair of correspondences $\langle e, f, = \rangle$ and $\langle e, g, = \rangle$:

If $|\{\langle e', f', = \rangle \in A; e' \sqsubseteq e \wedge f' \sqsubseteq f\}| > |\{\langle e', g', = \rangle \in A; e' \sqsubseteq e \wedge g' \sqsubseteq g\}|$, then $\langle e, g, = \rangle$ can be preserved; otherwise $\langle e, f, = \rangle$ is preserved.

In case the number of more specific matched entities are the same, then no correspondence can be selected. The same approach can be used with more general entities, e.g., superclasses, instead of subclasses. This technique applies for disambiguation of the same types of optimisation techniques used for matching (Sect. 6.1) or extracting alignments (Sect. 7.7.3).

7.8.2 Alignment Debugging

Alignment debugging or alignment repairing aims at restoring consistency and coherence of the produced alignment. Consistency is characterised by the aligned ontologies having no models and coherence by no model of the aligned ontologies allowing a particular class to have instances. The measure applied depends on the semantics given to alignments (see Sect. 2.5.3).

A comprehensive framework for alignment repair based on diagnosis theory was developed in (Meilicke and Stuckenschmidt 2009; Meilicke 2011). Minimal incoherence, or unsatisfiability, preserving subalignments (MIPS) are defined as minimal sets of correspondences generating inconsistency or incoherence, and diagnosis as an inclusion minimal subset of an alignment, whose removal would restore consistency or coherence. The relation between these concepts is that a minimal hitting set (taking one correspondence from each MIPS) is a diagnosis.

The notion of optimality of a diagnosis is then defined as the maximisation of the confidence in the repaired alignment. This optimisation can be applied for choosing the correspondence to suppress from each MIPS independently (locally optimal diagnosis) or globally (globally optimal diagnosis). In general, a globally optimal diagnosis is smaller than a locally optimal diagnosis.

Example 7.32 (Alignment debugging) Consider the alignment presented on Fig. 7.17.

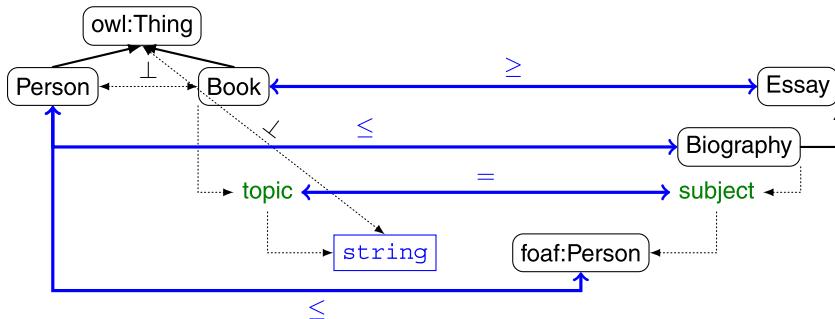


Fig. 7.17 An incoherent alignment. It is incoherent because the class Person cannot have any instance, otherwise it would be a Biography, thus a Book which is assumed to be disjoint from Person.

The alignment A contains four correspondences:

- $c_1 \quad \text{Book} \geq .8 \text{ Essay}$
- $c_2 \quad \text{Person} \leq .7 \text{ Biography}$
- $c_3 \quad \text{topic} = .6 \text{ subject}$
- $c_4 \quad \text{Person} \leq .9 \text{ foaf:Person}$

A system may use an incoherence pattern (or antipatterns; see Sect. 6.1.4), such as $\{X \perp Y, \langle X \leq Z \rangle, \langle Y \geq W \rangle, Z \sqsubseteq W\}$. Using the reduced semantics (Sect. 2.5.3), this pattern entails that X cannot have any extension, because otherwise this extension would be included in that of Y with which it is assumed to be disjoint. This pattern is instantiated in alignment A on the two correspondences c_2 and c_1 . A matcher using such a pattern could then decide to discard c_2 , because it has a lower confidence (.7).

However, this is not enough to render the alignment coherent. Indeed, the topic property can never have a value. This alignment contains two minimal unsatisfiable preserving subalignments (MUPS): $\{c_1, c_2\}$ and $\{c_1, c_3, c_4\}$. As a result, there exist three diagnoses: $\{c_1\}$, $\{c_2, c_3\}$ and $\{c_2, c_4\}$. A common practice would lead to reducing the impact of the change on the alignment and thus to retracting c_1 (globally optimal). However, a system may choose another diagnosis, such as $\{c_2, c_3\}$ (locally optimal) based on the average confidence or some other measure.

This alignment is also ambiguous because Person is matched to both Biography and foaf:Person. Disambiguating it by choosing the correspondence with the least confidence would discard c_2 .

Usage Examples The LogMap system (Sect. 8.3.26) uses a logical reasoner in order to pinpoint inconsistencies and incoherent classes. In order to scale to large ontologies, LogMap uses an incomplete reasoner. It selects the correspondences with the lowest confidence, from among the smallest sets of correspondences causing an inconsistency. The ASMOV system (Sect. 8.3.17) detects inconsistency through antipatterns and corrects the alignment before its final delivery. The inconsistency patterns used by ASMOV are semantically correct, but not complete, i.e., the remaining alignment may still be inconsistent. It then rejects some correspondences involved in these patterns, based on their confidence, and iterates the matching process. ContentMap (Sect. 8.4.7) can also be considered as a constraint-based debugging tool with the constraints provided by users.

The lack of disjointness axioms in ontologies prevents from applying ontology repair techniques through inconsistency detection. It is possible to use a naive Bayes classifier (Sect. 7.5.1) for learning how to generate disjointness axioms (Meilicke et al. 2008). Such a classifier is trained on various data sets and uses different similarity features (path distance, shared properties, similarity, instance sets) of pairs of classes, for deciding which ones are disjoint.

Techniques of this type may be applied to networks of ontologies as soon as they are given a semantics (see Sect. 2.5.3). Since repairing alignments at such a scale may be very demanding, both computationally and for users, (Zurawski et al. 2008) proposed restoring consistency only within spheres, which are local sets of ontologies and alignments.

ALCOMO⁶ (Applying Logical Constraints On Matching Ontologies) is the direct result of these works. Instead of presenting itself as an ontology matcher, it

⁶<http://web.informatik.uni-mannheim.de/alcomo/>.

provides a library of tools for computing MIPS, MUPS and diagnoses in either a complete or constraint-based way. It thus allows for implementing the measure and the selection parts of Fig. 7.16 independently from matchers. It also contains methods for generating upfront optimal incoherent avoidance alignments from a similarity matrix by pushing the coherence detection one step earlier in the matching process.

Summary on Alignment Improvement

Alignment improvement can be integrated within matchers or considered as a separate, post hoc, activity. We considered it separately because it is independent from matching and alignment extraction techniques and can be applied to alignments produced with any technique. Thus, any system may improve alignments based on such techniques. We only presented clearly specified techniques for improving alignments. As can be seen from the list of criteria given in the introduction to this section, other techniques can be developed in the same spirit.

7.9 Summary

We have presented the strategic issues involved in creating matching systems, besides using basic matchers of Chap. 5 or the more advanced global methods of Chap. 6. In particular, this involves composing basic matchers, aggregating their results and extracting alignments. This also covers the use of learning algorithms and strategies for improving extracted alignments.

The craft of ontology matching systems is a delicate art that combines basic matchers in the most advantageous way. This chapter has presented techniques used for assembling the components of a matching system. In most of the cases, the appropriate architecture depends on the problem to solve. Are there any independent basic matchers that can apply to the data? Is the data highly intricate? Are users available to evaluate the result? Must the expected result be injective? These questions are important components of a methodology (Chap. 3) and their answers lead to different assemblies of components.

With regard to the requirements of Chap. 1, the techniques presented in this chapter are often a matter of trade-off: between completeness and correctness of the alignment for threshold application, between quality and computation time for the choice of global similarity computation.

Figure 7.18 displays a fictitious example involving several methods. Specifically: (i) it runs several basic matchers in parallel, (ii) it aggregates their results, (iii) it selects some correspondences on the basis of their (dis)similarity, (iv) it extracts an alignment, (v) it repairs or disambiguates this alignment, and (vi) it iterates this process if necessary.

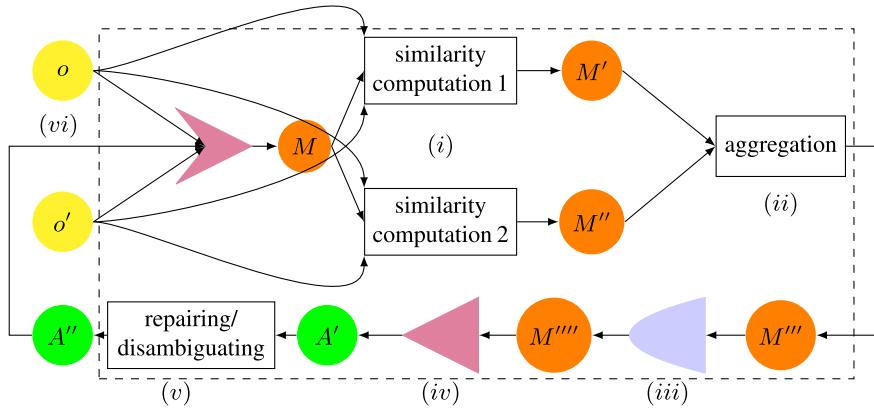


Fig. 7.18 A fictitious matching strategy.

The next part of the book describes how various implemented systems take advantage of the discussed techniques and how they compose them in a coherent system (Chap. 8). This will illustrate the diversity of approaches. We will then consider how to evaluate the merits of these systems experimentally (Chap. 9), because the mere theoretical consideration of a system capability and architecture is not a sufficiently convincing ground on which to judge its performances.

Part III

Systems and Evaluation

Chapter 8

Overview of Matching Systems

This chapter is an overview of matchers which have emerged during the last decades. There have already been some comparisons of matching systems, in particular in (Parent and Spaccapietra 2000; Rahm and Bernstein 2001; Do et al. 2002; Kalfoglou and Schorlemmer 2003b; Noy 2004a; Doan and Halevy 2005; Shvaiko and Euzenat 2005; Choi et al. 2006; Bellahsene et al. 2011). Our purpose here is not to compare them in full detail, though we give some comparisons, but rather to show their variety, in order to demonstrate in how many different ways the methods presented in the previous chapters have been practically exploited.

We have followed two principles in deciding whether a matching system should be included in this chapter: it must have an implementation and an archival publication describing it at the time of writing. We have also excluded from consideration the systems which assume that alignments have already been established, and use this assumption as a prerequisite of running the actual system. These approaches include such information integration systems as Tsimmis (Chawathe et al. 1994), Observer (Mena et al. 1996), SIMS (Arens et al. 1996), InfoSleuth (Fowler et al. 1999; Nodine et al. 2000), Kraft (Preece et al. 2000), Picsel (Goasdoué et al. 2000), DWQ (Calvanese et al. 2002a), AutoMed (Boyd et al. 2004), and InfoMix (Leone et al. 2005).

From around 50 systems covered in the first edition of the book, we added about 50 more new approaches, thus having in overall about 100 systems reviewed in this chapter (see Fig. 8.1 for a summary). This overview is still not exhaustive.

An interested reader can find a continuously updated and more complete information on the topic at [OntologyMatching.org](http://www.ontologymatching.org),¹ in particular, links to the web sites of the presented systems can be found there. We only mention addresses of general-purpose resources.

We present the matching systems in light of the classifications discussed in Chap. 4. We also point to concrete basic and advanced matchers and matching strategies used in the considered systems by referencing to the corresponding subsections

¹<http://www.ontologymatching.org>.

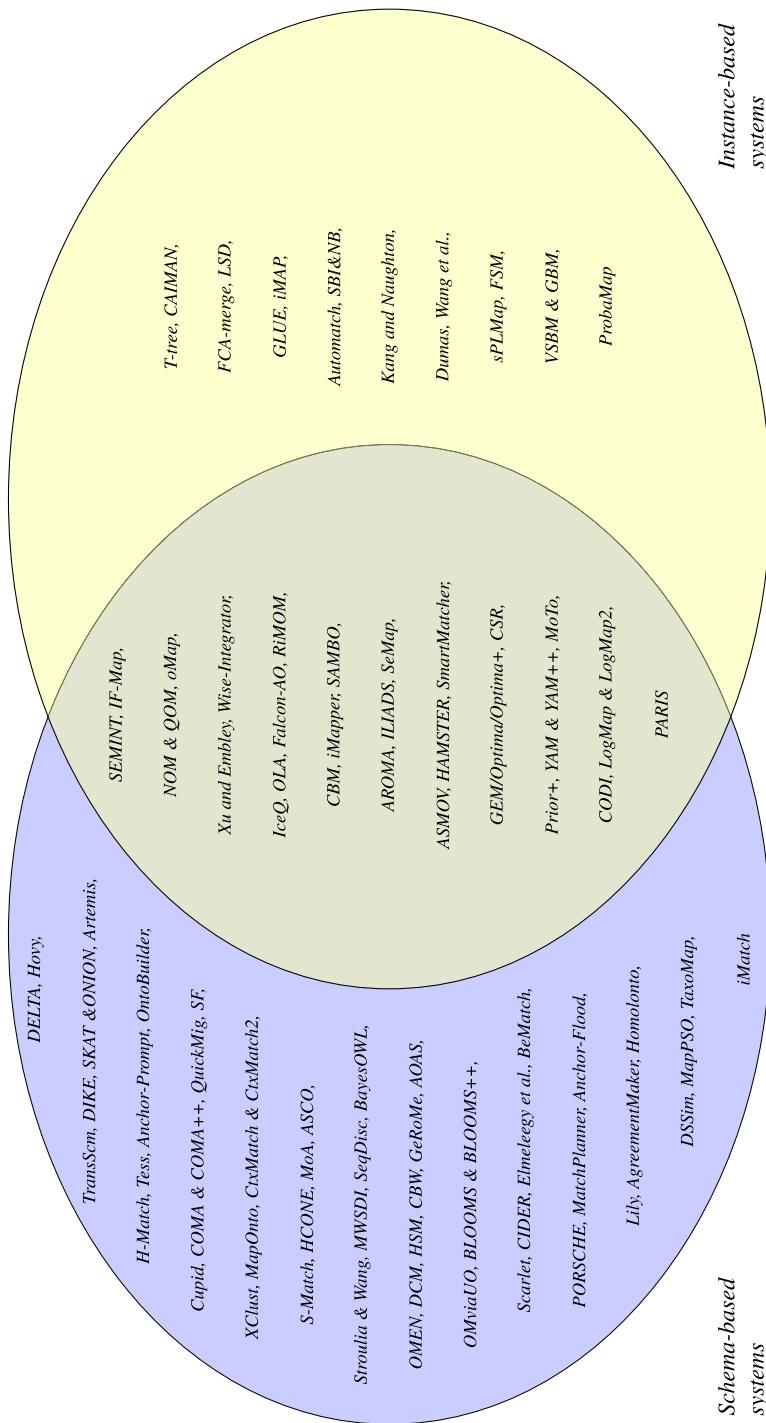


Fig. 8.1 Summary of the covered matching systems: 47 schema-based, such as Delta and iMatch, 15 instance-based, such as T-tree and ProbaMap, 27 mixed systems, such as SEMINT and PARIS. The chapter also covers 10 metamatching systems (not shown on the figure).

of Chap. 5, Chap. 6 and Chap. 7. In order to facilitate the presentation we follow two rules. First, the year of the system appearance is considered. Then, evolutions of the same system or very similar systems are discussed close to each other. We illustrate systems with the help of figures, if the matching process is of a particular interest or substantially complex. We tried to adopt a unified presentation for these systems. However, some of these are specific enough so that we did not enforce the terminology of Sect. 2.4 but kept it as used by system designers.

The structure of this chapter is as follows. We first describe systems which mostly focus on schema-level information (Sect. 8.1). Secondly, we discuss systems which concentrate on instance-level information (Sect. 8.2). Then, we present systems which exploit both schema-level and instance-level information (Sect. 8.3). Finally, we overview metamodeling systems (Sect. 8.4).

8.1 Schema-Based Systems

Schema-based systems, according to the classification of Chap. 4, are those which rely mostly on schema-level input information for performing ontology matching.

8.1.1 *DELTA (The MITRE Corporation)*

DELTA (Data Element Tool-based Analysis) is a system that semi-automatically discovers attribute correspondences among database schemas (Clifton et al. 1997). It handles relational and extended entity–relationship (EER) schemas. The idea of the approach is to use textual similarities between data element definitions in order to find matching candidates. The system converts available information about an attribute, e.g., attribute name, data type, narrative description, into a simple text string, called *document*. The documents describing each database attribute constitute a *document base*. Then, DELTA feeds the document base from the first schema into a full-text information retrieval tool, such as Personal Librarian. Matching is viewed as a Personal Librarian query based on the information from the second schema. The query can be a string of disconnected phrases, a full Boolean query, a few relevant words, or an entire document. The tool estimates the similarity (by using natural language heuristics, such as considering that rare or repeated words are more important) between a search pattern and contents of a document base (Sect. 5.2.1). It is thus exclusively based on string-based techniques. It returns a ranked list of documents that it considers to be similar. The selection of the final alignment is to be performed by users.

8.1.2 *Hovy (University of Southern California)*

(Hovy 1998) describes heuristics used to match large-scale ontologies, such as Sensus and Mikrokosmos, in order to combine them in a single reference ontology. In

particular, three types of matchers were used based on (i) concept names, (ii) concept definitions, and (iii) taxonomy structure. For example, the name matcher splits composite-word names into separate words (Sect. 5.2.2) and then compares substrings in order to produce a similarity score. Specifically, the name matcher score is computed as the sum of the square of the number of letters matched, plus 20 points if words are exactly equal or 10 points if end of match coincides. For instance, using this strategy, the comparison between Free World and World results in 35 points, while the comparison between cuisine and vine results in 19 points. The definition matcher compares the English definitions of two concepts (Sect. 5.2.2). Here, both definitions are first split into individual words. Then, the number and the ratio of shared words in two definitions is computed in order to determine the similarity between them. Finally, results of all the matchers are combined based on experimentally obtained formulas. The combined scores between concepts from two ontologies are sorted in descending order and are presented to users for establishing a cutoff value as well as for approving or discarding operations, results of which are saved for later reuse.

8.1.3 TransScm (*Tel Aviv University*)

TransScm (Milo and Zohar 1998) provides data translation and conversion mechanisms between input schemas based on schema matching. First, by using rules, an alignment is produced in a semi-automatic way. Then, this alignment is used to translate data instances of the source schema to instances of the target schema. Input schemas are internally encoded as labelled graphs, where some of the nodes may be ordered. Nodes of the graph represent schema elements, while edges stand for the relations between schema elements or their components. Matching is performed between nodes of the graphs in a top-down and one-to-one fashion. Matchers are viewed as rules (Zohar 1997). For example, according to the *identical* rule, two nodes match if their labels are found to be synonyms based on the built-in thesaurus (Sect. 5.2.2). The system combines rules sequentially based on their priorities. It tries to find, for the source node, a unique *best* matching target node, or determines a mismatch. In case there are several matching candidates among which the system cannot choose the best one, or if the system cannot match or mismatch a source node to a target node with the given set of rules, user involvement is required. In particular, users with the help of a graphic user interface can add, disable or modify rules to obtain the desired matching result. Then, instances of the source schema are translated to instances of the target schema according to the match rules. For the example of the *identical* rule, translation includes copying the source node components.

8.1.4 DIKE (*University of Reggio Calabria and University of Calabria*)

DIKE (Database Intensional Knowledge Extractor) is a system supporting the semi-automatic construction of cooperative information systems (CIS) from heterogeneous databases (Palopoli et al. 2003a, 2003b). It takes as input a set of databases belonging to the CIS. It builds a kind of mediated schema (called data repository or global structured dictionary) in order to provide a user-friendly integrated access to the available data sources. DIKE focusses on entity–relationship schemas. The matching step is called extraction of inter-schema knowledge. It is performed in a semi-automatic way. Some examples of inter-schema properties that DIKE can find are terminological properties, such as synonyms, homonyms among objects, namely entities and relationships, or type conflicts, e.g., similarities between different types of objects, such as entities, attributes, relationships; structural properties, such as object inclusion; subschema similarities, such as similarities between schema fragments. With each kind of property is associated a plausibility coefficient in the [0 1] range. The properties with a lower plausibility coefficient than a dynamically derived threshold are discarded, whereas others are accepted. DIKE works by computing sequentially the above mentioned properties. For example, synonyms and homonyms are determined based on information from external resources, such as WordNet (Sect. 5.2.2), and by analysing the distances of objects in the neighbourhood of the objects under consideration (Sect. 6.1). Some weights are also used to produce a final coefficient. Then, type conflicts are analysed and resolved by taking as input the results of synonym and hyponym analysis.

8.1.5 SKAT and ONION (*Stanford University*)

SKAT (Semantic Knowledge Articulation Tool) is a rule-based system that semi-automatically discovers mappings between two ontologies (Mitra et al. 1999). Internally, input ontologies are encoded as graphs. Rules are provided by domain experts and are encoded in first order logic. In particular, experts specify initially desired matches and mismatches. For example, a rule President ≡ Chancellor, indicates that we want President to be an appropriate match for Chancellor. Apart from declarative rules, experts can specify matching procedures used to generate the new matches. Experts have to approve or reject the automatically suggested matches, thereby producing the resulting alignment. Matching procedures are applied sequentially. Some examples of these procedures are: string-based matching, e.g., two terms match if they are spelled similarly (Sect. 5.2.1), and structure matching, e.g., structural graph slices matching, such as considering nodes near the root of the first ontology against nodes near the root of the second ontology (Sect. 6.1).

ONION (ONtology compositION) is a successor to SKAT that semi-automatically discovers mappings between multiple ontologies, in order to enable a unified query answering over these ontologies (Mitra et al. 2000). Input ontologies,

RDF files, are internally represented as labelled graphs. The alignment is viewed as a set of *articulation rules*. The semi-automated algorithm for resolving the terminological heterogeneity of (Mitra and Wiederhold 2002) forms the basis of the *articulation generator*, ArtGen, for the ONION system. ArtGen, in turn, can be viewed as an evolution of the SKAT system with some added matchers. Thus, it executes a set of matchers and suggests articulation rules to users. Users can either accept, modify or delete the suggestions. They can also indicate the new matches that the articulation generator may have missed. ArtGen works sequentially, first by performing linguistic matching (Sect. 5.2.2) and then structure-based matching (Sect. 5.3). During the linguistic matching phase, concept names are represented as sets of words. The linguistic matcher compares all possible pairs of words from any two concepts of both ontologies and assigns a similarity score in [0 1] to each pair. The matcher uses a word similarity table generated by a thesaurus-based or corpus-based matcher called the *word relator* to determine the similarity between pairs of words (Sect. 5.2.2). The similarity score between two concepts is the average of the similarity scores (ignoring scores of zero) of all possible pairs of words in their names. If this score is higher than a given threshold, ArtGen generates a match candidate. Structure-based matching is performed based on the results of the linguistic matching. It looks for structural isomorphism between subgraphs of the ontologies, taking into account some linguistic clues (see Sect. 8.1.11 for a similar technique). The structural matcher tries to match only the unmatched pairs from the linguistic matching, thereby complementing its results.

8.1.6 Artemis (*University of Milan and University of Modena e Reggio Emilia*)

Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) (Castano et al. 2000) was designed as a module of the MOMIS mediator system (Bergamaschi et al. 1998, 1999) for creating global views. It performs affinity-based analysis and hierarchical clustering of database schema elements. Affinity-based analysis represents the matching step: in a sequential manner it calculates the name, structural and global affinity coefficients exploiting a common thesaurus. The common thesaurus is built with the help of ODB-Tools (Beneventano et al. 1998), WordNet (Sect. 5.2.2) or manual input. It represents intensional and extensional relationships which depict intra- and inter-schema knowledge about classes and attributes of the input schemas. Based on global affinity coefficients, a hierarchical clustering technique categorises classes into groups at different levels of affinity. For each cluster, it creates a set of global attributes and the global class. Logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table.

8.1.7 H-Match (University of Milan)

H-Match (Castano et al. 2006) is an automated ontology matching system. It has been designed to enable knowledge discovery and sharing in the settings of open networked systems, in particular within the Helios peer-to-peer framework (Castano et al. 2005). The system handles ontologies specified in OWL. Internally, these are encoded as graphs using the H-model representation (Castano et al. 2005). H-Match inputs two ontologies and outputs (one-to-one or one-to-many) correspondences between concepts of these ontologies with the same or closest intended meaning. The approach is based on a similarity analysis through affinity metrics, e.g., term-to-term affinity, data type compatibility (Sect. 5.3.2), and thresholds. H-Match computes two types of affinities (in the [0 1] range), namely *linguistic* and *contextual* affinity. These are then combined by using weighting schemas, thus yielding a final measure, called *semantic* affinity. Linguistic affinity builds on top of the thesaurus-based approach of the Artemis system (Sect. 8.1.6). In particular, it extends the Artemis approach (i) by building a common thesaurus involving relations among WordNet synsets such as meronymy and coordinate terms, and (ii) by providing an automatic handler of compound terms, i.e., those composed by more than one token, that are not available from WordNet. Contextual affinity requires consideration of the neighbour concepts, e.g., linked via taxonomical or mereological relations, of the actual concept (Sect. 6.1).

One of the major characteristics of H-Match is that it can be dynamically configured for adaptation to a particular matching task, because in dynamic settings, the complexity of a matching task is not known in advance. This is achieved by means of four matching models. These are: *surface*, *shallow*, *deep*, and *intensive*, each of which involves different types of constructs of the ontology (see Fig. 8.2).

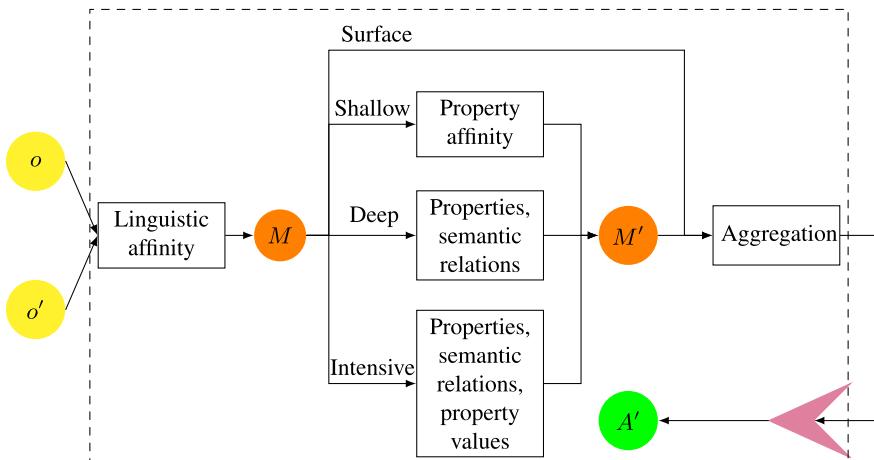


Fig. 8.2 H-Match matching process: H-Match is a conditional system that can use alternatively or in parallel four matching models depending on the resources available.

Computation of a linguistic affinity is a common part of all the matching models. In the surface model, linguistic affinity is also the final affinity, since this model considers only names of ontology concepts. All the other three models take into account various contextual features and therefore contribute to the contextual affinity. For example, the shallow model takes into account concept properties, whereas the deep and the intensive models extend previous models by including relations and property values, respectively. Each concept involved in a matching task can be processed according to its own model, independently from the models applied to the other concepts within the same task. Finally, correspondences with semantic (final) affinity higher than a cut-off threshold value are returned in the final alignment.

8.1.8 Tess (*University of Massachusetts*)

Tess (Type Evolution Software System) is a system to support schema evolution (see Sect. 12.8) by matching the old and the new versions (Lerner 2000). Schemas are viewed as collections of types. Since in the given application scenario changes are typically evolutionary, rather than revolutionary, it is assumed that input schemas are highly similar. Matching is viewed as generation of *derivation rules* to be applied to data. Tess can operate in modes ranging from fully automated to completely manual. Each derivation rule is associated with a similarity, which is meant to measure the impact that applying the derivation rule would have on existing data. By defining a threshold for the similarity, user involvement is determined. Matching is performed in three stages. First, the names of the types of old and new versions are compared (Sect. 5.2.1). Second, the structural information is taken into account. In particular, type constructors used by the old and new types and the types of components are analysed (Sect. 5.3.2). This provides the ability to handle cases in which, for example, component names have been changed, but their types are unchanged. Third, if everything else fails, matching relies upon some ordering information heuristics. Thus, in this case, Tess will try matching components with different names and different types. Finally, based on the derivation rules a transformer is produced which can update data in a database according to a newer version of the schema. In the simplest case, such as the identity derivation rule case, when type names are identical, as in TransScm (Sect. 8.1.3), the derivation function simply copies existing objects. A more complex transformation may include a join operation to combine two related objects into one.

8.1.9 Anchor-Prompt (*Stanford Medical Informatics*)

Anchor-Prompt (Noy and Musen 2001) is an extension of Prompt, also formerly known as SMART. It is an ontology merging and alignment tool with a sophis-

ticated prompt mechanism for possible matching terms (Noy and Musen 1999). Prompt handles ontologies expressed in such knowledge representation formalisms as OWL and RDF Schema. Anchor-Prompt is a sequential matching algorithm that takes as input two ontologies, internally represented as graphs and a set of anchors, which are identified with the help of string-based techniques, such as edit distance (Sect. 5.2.1), user-defined distance or another matcher computing linguistic similarity. Then the algorithm refines them by analysing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar positions on similar paths (Sect. 6.1.1). Finally, based on the frequencies and user feedback, the algorithm determines matching candidates.

The Prompt and Anchor-Prompt systems have also contributed to the design of other algorithms, such as PromptDiff (Sect. 10.3.5) which finds differences between two ontologies and provides the editing facility for transforming one ontology into another.

8.1.10 *OntoBuilder (Technion Israel Institute of Technology)*

OntoBuilder is a system for information seeking on the web (Modica et al. 2001). A typical situation the system deals with is when users are seeking for a car to be rented. Obviously, they would like to compare prices from multiple providers in order to make an informed decision. OntoBuilder operates in two phases: (i) ontology creation (the *training* phase) and (ii) ontology adaptation (the *adaptation* phase). During the training phase an initial ontology (in which user data needs are encoded) is created by extracting it from a visited web site of, e.g., some car rental company. The adaptation phase includes on-the-fly match and interactive merge operations of the related ontologies with the actual (initial) ontology. We concentrate below only on the ontology adaptation phase. During the adaptation phase, users suggest the web sites they would like to further explore, e.g., the ones of various car rental companies. Each such site goes through the ontology extraction process. This results in a candidate ontology, which is then merged into the actual ontology. To support this, the best match for each existing term in the actual ontology to terms from the candidate ontology is selected. The selection strategy employs thresholds (Sect. 7.7.1). The matching algorithm works in a term-to-term fashion, sequentially executing various matchers. Some examples of used matchers are substring matching (Sect. 5.2.1), or thesaurus look-up (Sect. 5.2.2). Finally, mismatched terms are presented to users for manual matching. Some further matchers, such as those for precedence matching, were introduced in later work (Gal et al. 2005b). Top- k mappings (Gal 2006) have been proposed as an alternative for a single best matching, i.e., top-1 category.

8.1.11 Cupid (University of Washington, Microsoft Corporation and University of Leipzig)

Cupid (Madhavan et al. 2001) implements an algorithm comprising linguistic and structural schema matching techniques and computing similarity coefficients with the assistance of domain-specific thesauri. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases (see Fig. 8.3) and operates only with tree structures to which nontree cases are reduced.

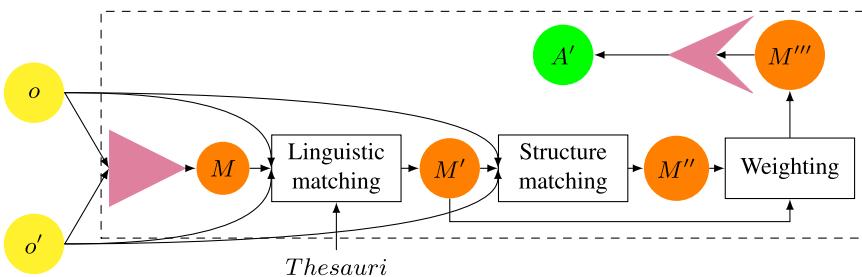


Fig. 8.3 Cupid architecture: it is a very common architecture which mixes parallel and sequential composition. Structure matching takes advantage of the results of linguistic matching, but the results of both of them are taken into consideration for weighting.

The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalisation (Sect. 5.2.2), categorisation, string-based techniques, such as common prefix, suffix tests (Sect. 5.2.1), and thesaurus look-up (Sect. 5.2.2). The second phase (structural matching) computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur (Sect. 6.1.3). The third phase (mapping elements generation) aggregates the results of linguistic and structural matching through a weighted sum (Sect. 7.4.1) and generates a final alignment by choosing pairs of schema elements with weighted similarity coefficients, which are higher than a threshold (Sect. 7.7.1).

8.1.12 COMA and COMA++ (University of Leipzig)

COMA (COmbination of MAtching algorithms) (Do and Rahm 2002) is a schema matching tool based on parallel composition of matchers. It provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for the evaluation of the different matchers. As from (Do and Rahm 2002), COMA contains 6 elementary matchers, 5 hybrid matchers, and a reuse-oriented matcher. Most of them implement string-based techniques, such as affix, n -gram,

edit distance (Sect. 5.2.1); others share techniques with Cupid, e.g., thesaurus look-up. An original component, called *reuse-oriented matcher*, tries to reuse previously obtained results for entire new schemas or for their fragments. Schemas are internally encoded as directed acyclic graphs, where elements are the paths. This aims at capturing contexts in which elements occur. Distinct features of the COMA tool with respect to Cupid are a more flexible architecture and the possibility of performing iterations in the matching process. It presumes interaction with users who approve obtained matches and mismatches to gradually refine and improve its matching accuracy. COMA++ is built on top of COMA by elaborating in more detail the alignment reuse operation. It also provides a more efficient implementation of the COMA algorithms and a graphical user interface (Sect. 10.3.2).

8.1.13 QuickMig (SAP, University of Leipzig)

QuickMig is a semi-automatic schema matching system targeting data migration applications (Drumm et al. 2007). It exploits sample instances, domain ontologies and reuse matchers to compute correspondences and the respective mapping expressions in order to transform instance data from a source schema to a target schema. The overall QuickMig migration process is organised in five steps:

- *Answering a questionnaire*, which is done manually by a user. By answering simple questions, users indicate fragments of the target schema, that are (ir)relevant for particular data migration needs. This is aimed at reducing the complexity of the target schema, and hence, of the matching task.
- *Injection of sample instances*, which is done manually by a user, who creates in the source system the sample instances already existing in the target system. It is assumed that creating the same instances in the source and the legacy systems can be done quickly by business users, since this is part of their daily activity.
- *Import* of the schemas to be matched as well as of the respective sample instances into the matching system.
- *Matcher execution* that provides similarities between the source and the target schemas together with a proposal for a mapping category, such as move, split, concatenate, etc. Eleven mapping categories were identified and are used to guide the creation of mapping expressions. For example, if the move category is assigned to a correspondence, this means that instance data from the source should be simply copied to the target schema element.
- *Review* is manually done by a user, who validates correspondences and completes mapping expressions. After that, the mapping code is generated and the resulting mappings are stored in a repository for future execution or reuse.

The QuickMig matching step is a further development of COMA/COMA++ (Sect. 8.1.12). Specifically, three instance-based matchers were added (Sect. 5.4) and its reuse matcher has been enhanced. These instance-based matchers are: *equality*, which identifies equal instance values in the source and the target, *split-concat*,

which verifies (by looking for substrings) if the instance data should be split or concatenated, and the *ontology-based matcher*, which exploits background knowledge codified in a domain ontology, e.g., modelling alternatives for common data structures such as telephone numbers. In turn, the reuse matcher was extended to handle the mapping categories mentioned above. The results of these matchers are combined through merging (with some cleaning).

8.1.14 Similarity Flooding (Stanford University and University of Leipzig)

The Similarity flooding approach (Melnik et al. 2002) is based on the ideas of similarity propagation. Schemas are presented as directed labelled graphs grounded on the OIM specification (MDC 1999). The algorithm manipulates them in an iterative computation to produce an alignment between the nodes of the input graphs. The technique starts from string-based comparison, such as common prefix, suffix tests (Sect. 5.2.1), of the vertices labels to obtain an initial alignment which is refined through iterative computation. The basic concept behind the Similarity flooding algorithm is the similarity spreading from similar nodes to the adjacent neighbours through propagation coefficients. From iteration to iteration the similarity measure is spread to the graph until a fixed point is reached or the computation is stopped. The full process is described in Sect. 6.2.1. The result of this step is an alignment which is further filtered to produce the final alignment.

8.1.15 XClust (National University of Singapore)

XClust is a tool for integrating multiple DTDs (Lee et al. 2002). Its integration strategy is based on clustering. Given multiple DTDs, it clusters them according to their similarity. This aims at facilitating the work of system integrators by allowing them to focus on already similar DTDs of single clusters. Clustering is applied recursively until a manageable number of DTDs is obtained. XClust works in two phases: (i) DTD similarity computation, and (ii) DTD clustering. During the first phase, given a set of DTDs, pairwise similarities between their underlying labelled trees are computed. This is done by using several matchers which exploit schema names as well as some structural information. For example, the *basic similarity* is computed as a weighted sum of a WordNet-based matcher that looks for synonyms among names of schema elements (Sect. 5.2.2) and a cardinality constraint matcher that performs a look-up in a cardinality compatibility table in order to compare cardinalities of schema elements (Sect. 5.3.4). Structural similarities exploit previously computed basic similarities and are based on (i) similarity of paths, (ii) similarity of immediate descendants and (iii) similarity of leaves (Sect. 6.1.3). For example, similarity of paths is computed as a normalised sum of basic similarities between

the sets of elements these paths are composed of, namely elements from the root to the node under consideration (Sect. 5.2.1). Structural similarities are aggregated as a weighted sum and then these aggregated similarities are used to choose the best match pairs by applying a threshold. These constitute the alignment for a pair of DTDs. Finally, for two DTDs, best match pairs are summed up and normalised, thereby resulting in a final similarity between these DTDs. The result of the first phase is the similarity matrix of a set of DTDs. During the second phase, based on the DTD similarity matrix, a hierarchical clustering (Everitt 1993) is applied to group DTDs into clusters.

8.1.16 MapOnto (*University of Toronto and Rutgers University*)

MapOnto is a system for constructing complex mappings between ontologies and relational or XML schemas (An et al. 2005, 2006). It takes as input three arguments: (i) an ontology specified in an ontology representation language, e.g., OWL, (ii) a relational or XML schema, and (iii) simple correspondences, e.g., between XML attributes and ontology data type properties. Input schema and ontology are internally encoded as labelled graphs. Then, the approach looks for ‘reasonable’ connections among the graphs. The system produces in a semi-automatic way a set of complex mapping formulas expressed in a subset of first-order logic (Horn clauses). The list of logical formulas is also ordered by the tool, thereby suggesting the most reasonable mappings. Finally, users can inspect that list and choose the best ones.

8.1.17 CtxMatch and CtxMatch2 (*University of Trento and ITC-IRST*)

CtxMatch (Bouquet et al. 2003a, 2003b) uses a semantic matching approach (Sect. 6.5.1). It translates the ontology matching problem into the logical validity problem and computes logical relations, such as equivalence, subsumption between concepts and properties. CtxMatch is a sequential system. At the element level it uses only WordNet to find initial matches for classes (Sect. 5.2.2). CtxMatch2 (Bouquet et al. 2006) improves on CtxMatch by handling properties. At the structure level, it exploits description logic reasoners, such as Pellet (Sirin et al. 2007) or FaCT (Tsarkov and Horrocks 2006) to compute the final alignment in a way similar to what is presented in Sect. 6.5.2.

8.1.18 S-Match (*University of Trento*)

S-Match implements the idea of semantic matching as initially described in (Giunchiglia and Shvaiko 2003). The first version of S-Match was a rationalised

re-implementation of CtxMatch with a few added functionalities (Giunchiglia et al. 2004). Later the system has undergone several evolutions, including extensions of libraries of element- and structure-level matchers, adding alignment explanations as well as iterative semantic matching (Giunchiglia et al. 2007, 2012a; Shvaiko et al. 2009). S-Match is limited to tree structures and does not consider properties or roles.

S-Match takes as input two graph structures, e.g., classifications, XML schemas, ontologies, and returns as output logic relations, e.g., equivalence, subsumption, which are supposed to hold between the nodes of the graphs. The relations are determined by (i) expressing the entities of the ontologies as logical formulas, and (ii) reducing the matching problem to a propositional validity problem. In particular, the entities are translated into propositional formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet. This allows for translating the matching problem into a propositional validity problem, which can then be efficiently resolved using (sound and complete) state-of-the-art propositional satisfiability solvers (see Sect. 6.5.1).

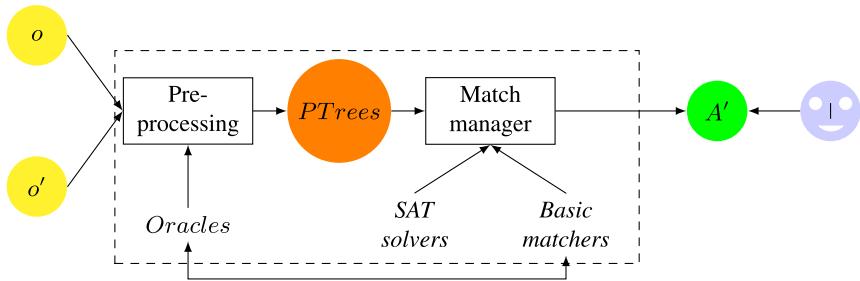


Fig. 8.4 S-Match architecture: ontology entities are converted to logic formulas by using the pre-processor and oracles. The Match manager then uses various basic element-level matchers and logic provers for finding relations between these formulas which, in turn, correspond to relations between entities. The computed alignments can be inspected with the help of a graphical user interface.

S-Match was designed and developed as a platform for semantic matching, namely a modular system with the core of computing semantic relations where single components can be plugged, unplugged or suitably customised. It is a sequential system with a parallel composition at the element level (see Fig. 8.4). The input ontologies are codified in a standard internal XML format. The module taking input ontologies performs some preprocessing with the help of oracles which provide the necessary a priori lexical and domain knowledge. Examples of oracles include WordNet (Sect. 5.2.2) and UMLS (Sect. 2.1.2). The output of the module is an enriched tree. These enriched trees are stored in an internal database (PTrees) where they can be browsed, edited and manipulated. The Match manager coordinates the matching process. S-Match libraries contain around 20 basic element-level matchers representing three categories, namely string-based, such as n -gram, edit distance

(Sect. 5.2.1), WordNet sense-based and WordNet gloss-based matchers (Sect. 5.2.2). Structure-level matchers include SAT solvers, e.g., those of SAT4J (Le Berre and Parrain 2010), and ad hoc reasoning methods (Giunchiglia et al. 2005b).

S-Match has been extended to reduce semantic heterogeneity in web service user descriptions (Giunchiglia et al. 2008). This has been evaluated with geo-web services (Vaccari et al. 2012). It discovers correspondences holding among the full tree structures and preserves certain structural properties of the trees under consideration. This extension is called a structure-preserving semantic matching operation (SPSM). This operation produces a set of correspondences between those nodes of the trees that correspond semantically to one another, (i) still preserving a set of structural properties of the trees being matched, namely that functions are matched to functions and variables to variables; and (ii) only in the case that the trees are globally similar to one another according to a similarity measure, which is based on a tree edit distance.

Another S-Match extension (Giunchiglia et al. 2012b) is minimal semantic matching. The key idea is that, based on a set of redundancy patterns, the minimal subset of correspondences can be computed such that all the other correspondences can be efficiently computed from them. It was shown that such an alignment always exists and it is unique. It is also possible to compute the set of maximum size, namely containing the maximum number of minimal and redundant correspondences, from the propagation of the correspondences in the minimal set.

8.1.19 HCONE (*University of the Aegean*)

HCONE is an approach to domain ontology matching and merging by exploiting different levels of interaction with users (Kotis et al. 2006; Vouros and Kotis 2005; Kotis and Vouros 2004). First, an alignment between two input ontologies is computed with the help of WordNet (Sect. 5.2.2). Then, the alignment is processed straightforwardly by using some merging rules, e.g., renaming, into a new merged ontology. The HCONE basic matching algorithm works in six steps:

1. Choose a concept from one ontology, denoted by c .
2. Obtain all the WordNet senses of c , denoted by s_1, s_2, \dots, s_m . For example, the concept Facility has five senses in WordNet.
3. Obtain hypernyms and hyponyms of all the senses of c (Sect. 5.2.2). For example, Police is a hyponym of Facility.
4. Build the $n \times m$ association matrix. This relates the n most frequently occurring terms in the vicinity of the m senses determined in step 2. The vicinity terms include those from the same synsets of m senses, hypernyms and hyponyms from step 3. In the case of the Facility example this is a 93×5 matrix. For example, the number of occurrences of such a vicinity term as Police is 3.
5. Build a query q by using subconcepts of c , e.g., TransportationSystem, or terms that are related to c via domain-specific relations in the input ontology. If the terms considered for q also exist among the n terms from step 4, then q memorises that

position with the help of flags. Thus, for the Facility concept, q is a 93 position vector, and, since the position of TransportationSystem is at the 35th place the value of $q[35]$ is 1.

6. Taking as input the association matrix computed at step 4 and the query computed at step 5, latent semantic indexing (Sect. 5.2.2) is used to compute the grades for what is the correct WordNet sense to be used for the given context (query).

The highest rated sense expresses the most plausible meaning for the concept under consideration. Finally, the relationship between concepts is computed. For instance, equivalence between two concepts holds if the same WordNet sense has been chosen for these concepts based on the procedure described above. The subsumption relation is computed between two concepts if a hypernym relation holds between the WordNet senses corresponding to these concepts. Based on the level at which users are involved in the matching process, HCONE provides fully automated, semi-automated and user-based algorithms to ontology matching. Users are involved in order to provide feedback on what is to be the correct WordNet sense on a one-by-one basis (user-based), or only in some limited number of cases, by exploiting some heuristics (semi-automated).

8.1.20 MoA (*Electronics and Telecommunication Research Institute, ETRI*)

MoA is an ontology merging and alignment tool (Kim et al. 2005). It consists of: (i) a library of methods for importing, matching, modifying, merging ontologies, and (ii) a shell for using these methods. MoA handles ontologies specified in OWL-DL. It is able to compute equivalence and subsumption relations between entities (classes, properties) of the input ontologies. The matching approach is based on concept (dis)similarity derived from linguistic clues. The MoA tool is a sequential solution. The preprocessing step includes three phases: (i) names of classes and properties are tokenised (Sect. 5.2.1); (ii) tokens of entities are associated with their meaning by using WordNet senses; (iii) meanings of tokens of ancestors of the entity under consideration are also taken into account, thereby extending the local meanings. This step is essentially the same as some part of the preprocessing done within S-Match (Sect. 8.1.18). Matching itself is based on rules. It is performed in a double loop over all the pairs of entities from the two input ontologies. For example, equivalence between two classes or properties holds when there is equivalence between these entities in either step (ii) or (iii). The equivalence, in turn, is decided via relations between the WordNet senses for one of the possible solutions (see Sect. 5.2.2). Thus, for example, author can be found to be equivalent to writer because they belong to the same synset in WordNet.

8.1.21 ASCO (INRIA Sophia-Antipolis)

ASCO is a system that automatically discovers pairs of corresponding elements in two input ontologies (Bach et al. 2004). ASCO handles ontologies in RDF Schema and computes alignments between classes, relations, and classes and relations. ASCO2, deals with OWL ontologies (Bach and Dieng-Kuntz 2005).

The matching process is organised sequentially in three phases. During the first phase (linguistic matching) the system normalises terms and expressions, e.g., by punctuation, upper cases, special symbols. Depending on their use in the ontology or if they are bags of words, ASCO uses different string comparison metrics for comparing the terms. Single terms are compared by using Jaro–Winkler, Levenshtein or Monge–Elkan (Sect. 5.2.1) and external resources, such as WordNet. Based on token similarities, the similarity between sets of tokens is computed using TFIDF. The obtained values are aggregated through a weighted sum.

The second phase (structure matching), computes similarities between classes and relations by propagating the input of linguistic similarities. The algorithm is an iterative fixed point computation algorithm that propagates similarity to the neighbours (subclasses, superclasses and siblings). Similarities between sets of objects are computed through single linkage. The propagation terminates when the class similarities and the relation similarities do not change after an iteration or a certain number of iterations is reached.

In the third phase, the linguistic and structural similarity are aggregated through a weighted sum and, if the similarities between matching candidates exceed a threshold (Sect. 7.7.1), they are selected for the resulting alignment.

8.1.22 Stroulia & Wang (University of Alberta)

Stroulia and Wang proposed a suite of methods for assessing similarity between two WSDL specifications in order to enhance automated service discovery (Stroulia and Wang 2005). Based on the textual description of a desired service, information retrieval methods are used to discover and order the matching service candidates. If a specification of the behaviour of the desired service is available, which can be partial, the matching candidates are further refined through structural similarity methods.

Specifically, the following methods have been proposed and implemented. When only service documentation (textual descriptions) are available, a traditional TFIDF vector-space method is used (Sect. 5.2.1). The vector-space method may be enhanced by also looking for synonyms, hyponyms, and hypernyms from WordNet (Sect. 5.2.2). The overall matching score is computed as the weighted average (Sect. 7.4.1) of the respective subvector matching scores. Different weights are applied giving importance, in descending order, to the original document terms, then to synonyms, and finally to hyponyms/hypernyms.

In turn, structure matching starts from comparing data types, then service messages and service operations. For example, data types are matched based on a compatibility look-up table made of four data type categories (Sect. 5.3.2). Messages are compared based on the similarity of their parameter lists in terms of the data types they contain and their organisation. These comparisons are performed in a pairwise fashion between source and target operations. The overall score is taken by maximising the sum of the individual scores. The final method directly extends structure matching, which looks essentially for data type compatibility and XML grouping styles, by considering also names of the respective service elements. This is performed based on WordNet by looking for synonyms or terms related hierarchically, e.g., as hyponyms or hypernyms. The system exhaustively explores possible correspondences between operations, and returns those with the highest matching scores. The overall similarity is computed as a sum of the names and the operation matching scores.

8.1.23 MWSDI (*University of Georgia*)

The METEOR-S Web Service Discovery Infrastructure (MWSDI) provides access to federations of service registries in a peer-to-peer network (Oundhakar et al. 2005). It is assumed that web services are annotated with concepts from different OWL ontologies. The web service discovery algorithm is based on matching search templates (ST), where users specify names, descriptions, operations, inputs, outputs of the desired web service, against candidate services (CS) available in service registries. Matching scores are computed for all the ST-CS pairs and ranked in descending order before being presented to users. The overall matching score is computed as a weighted average (Sect. 7.4.1) of service name, description and functional similarities. Name similarity is computed with the help of tokenisation (Sect. 5.2.2), n -grams (Sect. 5.2.1), and synonym look-up in WordNet (Sect. 5.2.2); description similarity is based only on n -grams. Functional similarity compares individual operation pairs, being a weighted average of name and description similarity, input and output similarity of the operations and concept similarity of the respective ontologies used to annotate the web services. The concept similarity is a weighted average of the name and description, property, and the coverage and context similarities. Name and description similarities are computed as described previously. Property similarity is based on name and description, range, cardinality as well as a penalty (reduction of 0.05) for unmatched properties. For example, data types in the range match are compared based on the compatibility look-up table (Sect. 5.3.2). Context similarity takes into account the neighbour concepts. In turn, coverage similarity determines whether concepts are semantically similar or disjoint and contributes accordingly to the matching score.

8.1.24 SeqDisc (*University of Leipzig, Queensland University of Technology, University of Magdeburg*)

SeqDisc is an automatic approach to WSDL service discovery (Algergawy et al. 2010). Web service specifications are viewed as rooted, labelled trees, which are divided into *concrete* and *abstract* parts to be further matched. The concrete part has a fixed hierarchical structure from the root (service name) to the portType element and various names. Abstract parts have differences in names and structures, as they correspond to the content of operation messages. The concrete hierarchical structures are matched by computing name similarity between elements at the same level only. This is done by combining tokenisation (Sect. 5.2.2), edit distance (Sect. 5.2.1) and 3-grams (Sect. 5.2.1). Abstract parts have differences in names and structures. These are first represented as Prüfer sequences (Prüfer 1918). Specifically, they are encoded as label Prüfer sequences (LPS) to capture information from labels and as number Prüfer sequences (NPS) to capture information from structures of web service operations. Then, similarities between service operation trees are computed by combining the string-based methods previously mentioned with data type comparison and children, leaf and ancestor structural similarities (Sect. 6.1.3). The matching result is refined through several alternatives, either by using only the results from the abstract tree comparison, or by using both abstract and concrete tree comparisons.

8.1.25 BayesOWL and BN Mapping (*University of Maryland*)

BayesOWL is a probabilistic framework for modelling uncertainty in the semantic web. It includes a Bayesian network mapping module (Sect. 6.4.1), which is in charge of automatic ontology matching (Pan et al. 2005). The approach works in three steps. First, two input ontologies are translated into two Bayesian networks. Specifically, classes are translated into nodes in a Bayesian network, while edges are created if the corresponding two classes are related by a *predicate* in the input ontologies. During the second step, matching candidates are generated between two Bayesian networks by learning joint probabilities from the web data. In particular, for each concept in an ontology, a group of sample text documents (called *examplars*) is created by querying a search engine. The query contains all the terms, e.g., {product book science} (opposed to a single term, e.g., {science}), in the path from the root to the concept (term) under consideration in the given ontology, thereby enabling some word sense disambiguation (Sect. 5.2.2). A text classifier, namely, Rainbow,² is trained on the statistical information about exemplars from the first ontology. Then, concepts of the second ontology are classified with respect to the concepts of the first ontology by feeding their exemplars to the trained classifier.

²<http://www.cs.umass.edu/~mccallum/bow/rainbow/>.

A similarity between two concepts is determined with the help of the Jaccard coefficient (Sect. 5.4) computed from the joint probabilities. These are used to construct conditional probability tables. During the third step, the mappings are refined as an update (combination of the Jeffrey rule and Iterative Proportional Fitting Procedure (Jeffrey 1983; Cramer 2000)) on probability distributions of concepts in the second Bayesian network, by distributions of concepts in the first Bayesian network, in accordance with the given similarities. By performing Bayesian inference with the updated distribution of the second Bayesian network, the final alignment is produced.

8.1.26 OMEN (*The Pennsylvania State University and Stanford University*)

OMEN (Ontology Mapping ENhancer (Mitra et al. 2005)) is a semi-automatic probabilistic ontology matching system based on a Bayesian network (Sect. 6.4.1). It takes as input two ontologies and an initial probability distribution derived, for instance, from basic (element-level) linguistic matchers. In turn, OMEN provides a structure-level matching algorithm, thereby deriving new mappings or discarding existing mappings. The approach can be summarised in four logical steps. First, it creates a Bayesian network, where a node stands for a mapping between pairs of classes or properties of the input ontologies. Edges represent the influences between the nodes of the network. This encoding is different from the one of BayesOWL (Sect. 8.1.25). During the second step, OMEN uses *metarules* that capture the influence of the structure of input ontologies in the neighbourhood of the input mappings in order to generate conditional probability tables for the given network. An example of a basic metarule is as follows. There are two conditions: (i) if the i -th concept from the first ontology, $c_{1,i} \in o_1$, matches the j -th concept from the second ontology, $c_{2,j} \in o_2$; (ii) if there is a relation q between concepts $c_{1,i}$ and $c_{1,k}$ in the first ontology, which matches a relation q' between concepts $c_{2,j}$ and $c_{2,m}$ in the second ontology. Then we can increase the probability of match between concepts $c_{1,k}$ and $c_{2,m}$. Other rules rely more heavily on the semantics of the language in which the input ontologies are encoded. During the third step, inferences are made (OMEN uses Bayesian Network tools in Java (BNJ)³) to generate *a posteriori* probabilities for each node. Finally, *a posteriori* probabilities, which are higher than a threshold (Sect. 7.7.1), are selected to generate the resulting alignment.

³<http://bnj.sourceforge.net>.

8.1.27 DCM Framework (*University of Illinois at Urbana-Champaign*)

MetaQuerier (Chang et al. 2005) is a middleware system that assists users in finding and querying multiple databases on the web. It exploits the Dual Correlation Mining (DCM) matching framework to facilitate source selection according to user search keywords (He and Chang 2006). Unlike other works, the given approach takes as input multiple schemas and returns alignments between all of them. This setting is called *holistic* schema matching. DCM automatically discovers complex correspondences, e.g., {author} corresponds to {first name, last name}, between attributes of web query interfaces in the same domain of interest, e.g., books. As the name DCM indicates, schema matching is viewed as *correlation mining*. The idea is that co-occurrence patterns often suggest complex matches. That is, *grouping attributes*, such as first name and last name, tend to co-occur in query interfaces. Technically, this means that those attributes are positively correlated. Contrarily, attribute names which are synonyms, e.g., quantity and amount, rarely co-occur, thus representing an example of negative correlation between them. Matching is performed in two phases. During the first phase (matching discovery), a set of matching candidates is generated by mining first positive and then negative correlations among attributes and attribute groups. Some thresholds and a specific correlation measure such as the H -measure are also used. During the second phase (matching construction), by applying ranking strategies, e.g., scoring function, rules, and selection, such as iterative greedy selection (Sect. 7.7.3), the final alignment is produced.

8.1.28 HSM (*Hong Kong University of Science and Technology, City University of Hong Kong*)

HSM is an approach to holistic schema matching among web query interfaces (Su et al. 2006). It computes 1:1, 1: n and m : n alignments and takes inspiration from DCM (Sect. 8.1.27), namely that correspondences can be discovered from term co-occurrence patterns within a domain. For example, in the book domain, the first name attribute is often co-present with the last name attribute, while it is rarely co-present with the author attribute. Frequent term co-presence indicates a *synonym* relation, while rare term co-presence indicates a *grouping* relation. The background intuition is that synonym attributes rarely appear in the same query interface, while grouping attributes usually appear together within the same query interface. Based on the above mentioned observations of the frequent or rare term co-presence, two scores are computed between every two attributes of all the schemas: (i) matching score evaluating the likeliness that two attributes are synonyms, (ii) grouping score evaluating the likeliness that two attributes are in the same group. Then, HSM employs a polynomial (with respect to the number of attributes) count-based greedy algorithm that iteratively selects the highest matching score to discover synonym correspondences. It also decides, based on the grouping score and a threshold, if two attributes that match the same set of other attributes belong to the same group.

8.1.29 CBW (*Sharif University of Technology, Tehran Institute for Studies in Theoretical Physics and Mathematics*)

CBW is a Coincidence-Based Weighting approach to ontology matching (Haeri (Hossein) et al. 2007; Qazvinian et al. 2008). It takes as input two OWL ontologies and a matrix of similarities between their concepts, which is computed through the Levenshtein distance (Sect. 5.2.1), and outputs a coincidence weighted alignment between these concepts. Ontologies are viewed as typed graphs, where concepts correspond to nodes, while relations correspond to typed edges. The coincidence measure (namely of being the same) is defined to evaluate coincidence of the graph structures. Specifically, it is used to weight candidate alignments and to extract the final one. This is done through evaluating partial orders on possible homomorphisms between graphs. The proposed partial order is a coincidence-based weighting function. The preservation of edges between corresponding nodes is considered ideal. Based on various combinations of edge preservation or non-preservation and node proximity, various weighting bonuses and penalties are defined to form this function. The approach specifically investigates several coincidence-based alignment extraction strategies (Sect. 7.7), such as:

- Optimal alignment through exhaustive search, which generates all possible alignments, scores them based on the coincidence measure and selects the one having the global maximal score;
- Near-optimal alignment, which adopts genetic algorithms (Sect. 7.6.2), where cross-over functions are based on the coincidence measure, such that new individuals have better coincidence in the new generations (Qazvinian et al. 2008);
- Approximate alignment extraction, which is based on random walks looking for a higher coincidence factor and maximum weight non-crossing matching (Malucci et al. 1993).

8.1.30 GeRoMeSuite (*RWTH Aachen University*)

GeRoMeSuite (GEneric ROle-based MEtamodel) is an environment for model management (Kensche et al. 2007b). It exploits role-based modelling to provide a unified generic representation for various models, such as relational, XML or OWL (Kensche et al. 2007a). It implements various model management operators (Sect. 10.3.1), such as match (Quix et al. 2007b), compose (Kensche et al. 2009), and merge (Quix et al. 2007a) that operate in a generic fashion, namely disregarding the original models. One of the intuitions behind this work is that different models can be encoded differently as labelled graphs, and then, these encodings influence particularly structural similarities among these graphs. This might become an issue especially if matching across heterogeneous models, such as XML and OWL, is needed.

GeRoMeSuite provides various graph traversals over the models, e.g., through namespaces, class hierarchies, or associations. It implements various element-level string-based matchers, such as Levenshtein, Jaro–Winkler, SMOA (Sect. 5.2.1) and a name path matcher (Sect. 5.2.1). The included structure-level matchers are Similarity flooding (Sect. 6.2.1) and the children matcher (Sect. 6.1.3). The system provides flexible ways to configure these matchers, e.g., by providing coefficients or number of iterations, as well as to aggregate, e.g., through average, and to filter, e.g., through thresholds, the matcher results with the help of a graphical user interface. The models to be matched are shown side by side in a tree view, while correspondences are displayed in the middle as coloured lines, thereby distinguishing between various degrees of similarity. Finally, GeRoMeSuite was also used together with SMB (Sect. 8.4.8), namely a metamatching system, in an integrated approach (Quix et al. 2010).

8.1.31 AOAS (US National Library of Medicine)

AOAS stands for the Anatomy Ontology Alignment System (Zhang and Bodenreider 2007), which as the title indicates was specifically designed to investigate alignment of anatomical ontologies. The system handles OWL ontologies and discovers 1:1, 1: m , and n : m alignments. The approach combines direct and indirect matching methods. Direct matching uses terminological techniques (exact and normalised equality) between concept names. Terms that are lexically similar are called anchors. The terminological matching is followed by the structural validation of these anchors through shared or compatible, though not identical, hierarchical (*is-a* and *part-of*) paths among concepts of the matched ontologies. In turn, the indirect matching between o and o' is performed via the reference ontology o_R , such as UMLS (Sect. 2.1.2). The evidence for the indirect match is the combination of evidences for two direct matches to the reference, i.e., between o and o_R and between o' and o_R . Finally, the system combines (by using experimentally established weighting schemes) direct and indirect matching. The intuition behind AOAS is that the strongest correspondences are those supported by positive evidences, e.g., structural similarity, in both direct and indirect alignments. The presence of negative evidences, e.g., conflicting paths, in either case causes a correspondence to be discarded.

8.1.32 Scarlet (The Open University)

Scarlet is a system for SemantiC relAtion discoveRy by harvesting onLinE onToloGies (Sabou et al. 2008a, 2008b). It operates by contextualising the ontologies to be matched with ontologies that can be found on the web (see Sect. 7.3). The rationale behind Scarlet is that using more than one ontology improves the results. Scarlet operates along the following steps:

- *Harvesting ontologies* from the web: this is usually achieved through using Swoogle (Ding et al. 2005) or Watson (d’Aquin and Motta 2011).
- *Anchoring* the concepts to be matched to the appropriate part of these ontologies, i.e., a set of ontologies that can lead to a mapping. This is performed through exact string matching of concept lemmas.
- *Selecting ontologies* relying on the popularity-based mechanism of Swoogle, i.e., the first ontology it returns is inspected first; if no relation can be derived from it, then the next ontologies are considered.
- *Inferring relations* using rules that specify how a relation is computed, e.g., through direct and declared relations, as well as through transitivity of subsumption relations, to harness also indirect relations between concepts in the background knowledge.
- *Aggregating mappings* derived from the considered ontologies with the help of ad hoc inference rules to detect and handle the contradictory mappings, e.g., removing them.

Two strategies were investigated: (i) selecting for each pair of concepts an ontology in which they are both anchored and returning all the relations—or in a specific variant, the first relation—found between them; (ii) when no on-line ontology defines a relation for the previous case, multiple ontologies are investigated and relevant information from these is combined to derive a mapping. This latter variation can become a very complex procedure, so it is restricted to finding, for each pair of ontologies, the intersection between the entities subsuming one term and those subsumed by the other, which helps in quickly finding subsumption relations.

The Scarlet system was improved by combining it with CIDER (Sect. 8.1.35). This provided it with word sense disambiguation techniques, which allow for better discriminating between similar terms (Gracia et al. 2007). More recently, (Po and Bergamaschi 2010) proposed improving Scarlet through various word sense disambiguation algorithms, such as structural disambiguation, WordNet domain disambiguation, and heuristic rules, such as WordNet first sense rule, were used to provide annotations for each concept, i.e., to assign it an explicit sense.

8.1.33 OMviaUO (*University of Genova, Universidad Politécnica de Valencia*)

Upper-level ontologies may be exploited as a source of background knowledge in order to enhance automatic ontology matching between concepts (Sect. 7.3). OMviaUO (Mascardi et al. 2010) uses OpenCyc, SUMO-OWL and DOLCE (see Sect. 2.1.6) as such. It anchors the ontology to be matched with string-based, such as substring, *n*-gram and SMOA (Sect. 5.2.1), and language-based, such as based on WordNet (Sect. 5.2.2), basic matching methods. They are run in parallel (*parallel_match*) and their results are aggregated through the union of all the correspondences discovered by the single methods.

Several matching alternatives were investigated with these upper-level ontologies (*uo*) as input, such as:

- *uo_match*, which relies only on concept (*c*) names, and thus does not take into account structures of the input ontologies. Specifically, it uses *parallel_match* to match two input ontologies (*o* and *o'*) to an upper-level ontology (*u*) and then *composes* the resulting alignments. The intuition is that if both $c \in o$ and $c' \in o'$ match the same concept $c_u \in u$, then *c* and *c'* are related.
- *structural_uo_match*, similarly to the previous one, relies on *parallel_match* to match two input ontologies to an upper-level ontology. The difference is in the *structural composition* of the resulting alignments. It extends the intuition from the item above by also considering possible superconcept relations in the upper-level ontology. For example, if $c \in o$ matches $c_u \in u$, $c' \in o'$ matches $c'_u \in u$ and c'_u is a superconcept of c_u or if c_u and c'_u have a common superconcept, then *c* and *c'* are related.
- *mixed_match*, which aggregates results of *structural_uo_match* and direct *structural_parallel_match* is an extension of *parallel_match* with structural heuristics, such as verifying either if $c \in o$ is also similar to a superconcept of $c' \in o'$ or if $c \in o$ and $c' \in o'$ have similar respective superconcepts.

8.1.34 BLOOMS/BLOOMS+ (Wright State University, Accenture Technology Labs and Ontotext AD)

BLOOMS (Bootstrapping-based Linked Open Data Ontology Matching System) is a system for discovering correspondences or schema-level links across linked open data sets (Jain et al. 2010). The approach focusses on subsumption and equivalence relations between classes of the linked open data cloud ontologies. The key idea is that of bootstrapping from the community generated data available on the web. Specifically, Wikipedia and its category hierarchy, namely a user-generated categorisation of its pages, are used. The matching process is organised as follows:

- *Preprocessing of the input ontologies*, which removes properties and individuals and tokenises composite class names.
- *Construction of the BLOOMS forests*, namely of a set of trees, each of which corresponds to a Wikipedia sense (its disambiguation page). These trees are constructed for each class of the ontologies to be matched by using Wikipedia categories (by taking a concise selection of supercategories for the class name under consideration).
- *Forest comparison*, which results in the first set of correspondences based on an ad hoc structural heuristic that counts the number of common nodes.
- *Postprocessing*, which is performed with the help of (i) the Alignment API (Sect. 10.3.6) to provide additional correspondences that pass a cut-off threshold of .95 and (ii) the Jena reasoner (Carroll et al. 2004) to infer further correspondences based on ontology structures and available alignments.

Subsequently the system evolved into a newer version, called BLOOMS+ (Jain et al. 2011), that reinforced the third step of the process described above. In particular, it uses a more sophisticated overlap similarity measure when comparing BLOOMS forests to determine matches between classes. It also computes a contextual similarity between classes, based on the taxonomic structure (Sect. 6.1.1) of ontologies, to obtain additional evidence in order to retain or to discard a correspondence. These two similarities are then aggregated through weighted average (Sect. 7.4.1).

8.1.35 CIDER (*Universidad Politécnica de Madrid, University of Zaragoza*)

CIDER is a Context and Inference baseD alignER (Gracia et al. 2011; Trillo et al. 2007). It handles OWL ontologies and computes equivalence correspondences between classes and properties. The matching process is organised in four steps as follows. First, for each ontology term, its context is extracted, namely by involving its synonyms, hypernyms, hyponyms, properties, domains, etc. This involves consulting WordNet or other ontologies indexed by Swoogle. Transitive inference is also used to include information not explicitly stated in the ontologies. Such context information is used to reduce the ambiguity of the terms to be matched (Trillo et al. 2007). Then, similarities between term labels are computed through the Levenshtein edit distance (Sect. 5.2.1), while structure (taxonomy and relational) similarities are computed through a vector space model by comparing features of the extracted contexts. At the third step, these individual similarities are adaptively combined with the help of an artificial neural network (Sect. 7.5.3). Specifically, a multi-layer perceptron is used. Finally, 1:1 term alignment extraction is performed through thresholds (Sect. 7.7.1). The system was developed by extending the Alignment API (Sect. 10.3.6).

8.1.36 Elmeleegy and Colleagues (*Purdue University*)

Elmeleegy and colleagues (Elmeleegy et al. 2008) take advantage of usage-based information, such as query logs, as additional schema (or instance) information in order to ground correspondence discovery between attributes of two schemas. The background intuition is to look for similarities in the query patterns to match attributes, which presumably play the same role in the schemas under consideration. Specifically, co-occurrence patterns between attributes as well as their use in joins and with aggregate functions are exploited. Several scoring functions, e.g., Euclidean distance, were considered to measure similarity between features of the attributes. A genetic algorithm (Sect. 7.6.2) was used to find the highest score corre-

spondences (based on scoring functions) between schemas. Two specific matchers were proposed: the structure-level usage-based matcher (SLUB) and the element-level usage-based matcher (ELUB).

The approach is organised in two main phases: (i) feature extraction and (ii) matching. During the first phase, *structure-level* features, namely those capturing the usage relationships, e.g., select-where, select-groupby, where-select, between attributes of the same schema, and *element-level* features, namely those capturing the usage of each attribute in isolation from the other attributes, e.g., occurrence in joins or with aggregate functions, such as count, minimum, maximum, sum or average are extracted from the query log of each schema. In turn, matching is performed through a genetic search, which examines several potential correspondences and assigns a score for each of them based on how well the features of the corresponding attributes match. A weighted average of the individual scores is used by the genetic algorithm as the fitness value. The approach was experimented with in combination with a data type matcher based on a data type compatibility matrix (Sect. 5.3.2) and with Similarity flooding (Sect. 6.2.1).

8.1.37 BeMatch (*Versailles Saint-Quentin en Yvelines, University of Cauca*)

BeMatch is a platform for matching and ranking service behaviour models (Corrales et al. 2008; Grigori et al. 2008). The system handles BPEL and WSCL protocols, provides a global similarity between these, an alignment as well as a script of edit operations required to transform the protocols under consideration. The approach operates on behaviour models (in contrast to service inputs and outputs) and besides exact matches, also delivers approximate matches. Users formulate requirements as process models, which are used as queries to retrieve services whose respective process models match with the whole or part of these queries. The service matching problem is reduced to a graph isomorphism problem (Sect. 6.1). Specifically, approximate matching is reduced to an error-correcting subgraph isomorphism problem. It is based on graph edit operations that, in turn, are used for determining a global distance between the query graph and the target graph. This is facilitated by: (i) a linguistic analyser to match node labels, using n -grams (Sect. 5.2.1), checking synonyms through WordNet (Sect. 5.2.2), etc.; (ii) a granularity analyser, which checks if the same functionality is modelled at different granularity levels in two graphs, e.g., as two nodes or as a single node; (iii) a cost function builder, which assigns costs to different graph edit operations reflecting the importance of dissimilarities between graphs, and allows for computing a global distance between graphs. Finally, services are ranked based on the global graph distance and cost.

8.1.38 PORSCHE (*University of Montpellier, ETH Zurich*)

PORSCHE is a Performance ORiented SCHEma mediation system (Saleem et al. 2008). It handles XML schemas and outputs 1:1, 1: m and n :1 alignments that are used to produce a mediated schema out of the input schemas. The approach follows a holistic approach by matching a set of XML schemas and by incrementally creating an integrated schema. Firstly, the method clusters the nodes in the given batch of schemas based on terminological label similarity (Sect. 5.2), e.g., token-level synonyms and abbreviation domain look-up tables, thereby producing clusters of nodes with similar labels. Then, it applies tree mining (which looks for subtree patterns that are frequent in the given set of trees) using node ranks calculated during depth-first traversal. This minimises the target search space (since the source node and candidate target nodes are in the same cluster; Sect. 7.1.2) and contributes to the scalability of the approach. The system is hybrid and extensible to incorporate more matching methods. Finally, it applies a binary ladder incremental schema match (Batini et al. 1986) and integrates techniques to produce the mediated schema together with mappings from source schemas to the mediated schema.

8.1.39 MatchPlanner (*University of Montpellier*)

MatchPlanner is a system implementing a planning approach to the combination of matching algorithms (Duchateau et al. 2008). It uses a decision tree (Sect. 7.5.5) as an aggregation function in which the nodes represent similarity measures or matchers and edges are used as conditions on the matcher results. Such a decision tree represents a plan, namely an ordered sequence of matching algorithms. A similarity computed by a matcher, which is a tree node, is evaluated against a condition, which is an edge. This controls the access to the subsequent tree nodes, and so, the execution of other matchers or terminates the process, if a leaf node is reached. Leaf nodes of these trees are either *true* or *false*, meaning that there either exists a match or not. For a given matching task the system uses only a subset (involving presumably the most appropriate one) of the matching algorithms from an extendable library of matchers. Thus, the complexity of the matching process is bounded by the depth of the decision tree. The system uses various string-based matchers, such as Levenshtein, 3-grams, Jaro–Winkler from the SecondString package (Sect. 5.2.1), WordNet-based matchers as well as a neighbour context matcher from (Duchateau et al. 2007b). The system is also self-configurable, namely it is able to select the optimal configuration of matchers by finding its most appropriate parameters, such as thresholds, weights, and coefficients (Sect. 7.6). Specifically, since edges in the decision tree are used as conditions, these can be viewed as thresholds, personalised to each matcher. Finally, decision trees can be designed manually or generated using machine learning (Sect. 7.5.5). These are often domain-specific, so they may require to be redesigned when a matching task from a new domain is considered.

8.1.40 Anchor-Flood (*Toyohashi University of Technology*)

The Anchor-Flood approach aims at handling efficiently particularly large ontologies (Hanif and Aono 2009). It inputs ontologies in RDFS and OWL and outputs 1:1 alignments. The system starts with a pair of similar concepts from two ontologies called an anchor, i.e., all pairs of concepts whose normalised names exactly match. Then, it gradually proceeds by analysing the neighbours, i.e., superconcepts, subconcepts, or siblings, of each anchor, thereby building small segments (fragments) out of the ontologies to be matched. The size of the segments is determined dynamically, starting from an anchor and exploring the neighbouring concepts until either all the collected concepts are explored or no new matching pairs are found. The system focusses on (local) segment-to-segment comparisons. Thus it does not consider the entire ontologies. This improves the system scalability (Sect. 7.1.2). It outputs a set of correspondences between concepts and properties of the semantically connected segments. For determining the correspondences between segments the approach relies on terminological (WordNet and Winkler-based string metrics; Sect. 5.2) and structural similarity measures, which are further aggregated by also considering probable misalignments. The similarity between two concepts is determined by the ratio of the number of terminologically similar direct superconcepts on the number of total direct superconcepts. Retrieved (local) matching pairs are considered as anchors for further processing. The process is repeated until there is no more matching pair to be processed.

8.1.41 Lily (*Southeast University, Nanjing University*)

Lily is an ontology matching system able to handle large ontologies and equipped with debugging facilities to verify and improve alignments (Wang et al. 2011; Wang and Xu 2009). It computes 1:1 alignments between concepts or properties. The matching process is organised in three steps. First, ontology entities are captured as ontology subgraphs through an extraction algorithm based on an electrical circuit model. Then, matching is performed over the ontology subgraphs. Lily combines various methods, namely string-based, such as Levenshtein distance (Sect. 5.2.1) between entity names and comments, and structure-based, such as a variation of Similarity flooding (Sect. 6.2.1) with various propagation strategies (Wang and Xu 2009). Large ontologies are handled based on reduction anchors (see Sect. 7.1.2). Finally, the last step of the matching process is debugging (Wang and Xu 2008). It aims at detecting deficiencies or errors, such as redundant, inconsistent, or imprecise correspondences in an alignment (Sect. 7.8.2). For example, a correspondence is redundant if it can be deduced from the existing ones. Some errors are repaired automatically, while cases treated as warnings are presented to users with suggestions for revision.

8.1.42 *AgreementMaker (University of Illinois at Chicago)*

AgreementMaker is a system offering a wide range of automatic matchers, an extensible and modular architecture, a metapurpose user interface, a set of evaluation strategies, and various manual, e.g., visual comparison, and semi-automatic features, e.g., user feedback (Cruz et al. 2009). It has been designed to handle large-scale ontologies based on the requirements coming from various domains, such as the geospatial and biomedical domains. The system handles ontologies in XML, RDFS, OWL, N3 and outputs 1:1, 1: m , n :1, and n : m alignments. In general, the matching process is organised into two modules: similarity computation and alignment selection. The system combines matchers using three layers:

- The matchers of the first layer compare concept features, such as labels, comments, and instances, which are represented as TFIDF vectors used with a cosine similarity metric (Sect. 5.2.1). Other string-based measures, e.g., edit and substring distances, may be used as well.
- The second layer uses structural ontology properties. It includes two matchers: descendant similarity inheritance, i.e., if two nodes are matched with high similarity, then the similarity between the descendants of those nodes should increase, and sibling similarity contribution, which uses the relationships between sibling concepts (Cruz and Sunna 2008).
- At the third layer, a linear weighted combination is computed over the results coming from the first two layers (Sect. 7.4.1), whose results are further pruned based on thresholds and desired output cardinalities of the correspondences (Sect. 7.7).

The system has a sophisticated user interface deeply integrated with the evaluation of ontology alignment quality. It empowers users with more control over the matching process.

8.1.43 *Homolonto (University of Lausanne, Swiss Institute of Bioinformatics)*

Homolonto is a system for matching anatomical ontologies (Parmentier et al. 2010). It uses the specifics of these ontologies, such as redundancy of terms, to discover and propagate similarity between concepts. When the similarity is important enough, concepts (anatomical elements) are considered as homologous and recorded as a possible correspondence. The system discovers many-to-many alignments. Homolonto uses weights for combining the results of various matchers (Sect. 7.4.1), such as those performing comparison of names (identical strings, synonyms), statistics of word occurrences, graph structure comparison, or user input. Specifically, first word matching of concept names and possibly user input are used to produce initial propositions, also called anchors. Then, the scores from these initial propositions are propagated to the neighbours (children). The system is equipped with a

graphical user interface. Propositions are displayed to users in descending order for validation. Users can accept a correspondence as homology, as partial homology, reject it or postpone the decision. After user validation, word matching and structure matching are iterated until users decide to terminate the process or no new propositions are generated. Since homology relationships are transitive, Homolonto allows for merging such correspondences into homologous (organ) groups, which in turn may be used for building a multispecies ontology.

8.1.44 DSSim (Open University, Poznan University of Economics)

DSSim is an agent-based ontology matching framework. The system handles large-scale ontologies in OWL and SKOS and computes 1:1 alignments with equivalence and subsumption relations between concepts and properties. It uses the Dempster-Shafer theory (Sect. 7.4.2) in the context of query answering (Nagy and Vargas-Vera 2010). Specifically, each agent builds a belief for the correctness of a particular correspondence hypothesis. Then, these beliefs are combined into a single more coherent view in order to improve correspondence quality. The ontologies are initially partitioned into fragments. Each concept or property of a first ontology fragment is viewed as a query, which is expanded based on hypernyms from WordNet. These hypernyms are used as variables in the hypothesis to enhance the beliefs. The expanded concepts and properties are matched syntactically to the similar concepts and properties of the second ontology in order to identify a relevant graph fragment of the second ontology. Then, the query graph of the first ontology is matched against the relevant graph fragment of the second ontology. For that purpose, various terminological similarity measures are used, such as Monge–Elkan and Jaccard similarity, which are combined using the Dempster rule (Sect. 7.4.2). Similarities are viewed as different experts in the evidence theory and are used to assess quantitative similarity values (converted into belief mass functions) that populate the similarity matrices. The resulting correspondences are selected based on the highest belief function over the combined evidences. Conflicts among beliefs are resolved by using a fuzzy voting approach equipped with four ad hoc *if-then* rules.

8.1.45 MapPSO (FZI Research Center for Information Technology, Griffith University)

MapPSO is an approach to Ontology Matching based on discrete Particle Swarm Optimisation (Bock et al. 2011; Bock and Hettenhausen 2012). In such a setting, the matching problem is reduced to the biologically inspired non-deterministic population-based optimisation where the objective function of alignment quality is optimised (Sect. 6.3.2). The system handles OWL ontologies and computes 1:1

alignments between concepts and properties. The process starts with a random alignment, which is then evaluated and updated iteratively. Evaluation (fitness) is performed through a combination of basic matchers, such as the SMOA string distance, WordNet-based distance or hierarchy distances as implemented in OntoSim (Sect. 5.2.1). Various aggregations can be applied, such as weighted average or ordered weighted average (Sect. 7.4.1). The approach maintains a constant population of individuals or swarm particles, viewed as candidate alignments. These are updated iteratively by changing their positions in the search space based on particle memory and communication between particles in order to converge to an optimal alignment. Each particle can be updated in parallel with other particles.

Recently, a variation of this population-based optimisation approach was proposed under the name of MapEVO (Bock et al. 2011). Instead of particle swarm optimisation, it is based on evolutionary programming (Sect. 7.6.2). MapEVO optimises the same objective function of alignment quality as MapPSO, though populations of candidate alignments (or individuals) are viewed as species that evolve, i.e., become extinct or reproduce themselves.

8.1.46 TaxoMap (University of Paris-Sud 11, INRIA)

TaxoMap is a system for discovering equivalence, subsumption and proximity (*is-Close*) correspondences among taxonomies (Hamdi et al. 2010a, 2010b). Matching is based only on labels of concepts and subsumption hierarchies. TaxoMap starts by classifying words occurring in labels as *full* or *complimentary*, based on their part-of-speech and position in the labels. This is performed with the help of TreeTagger.⁴ Then, these are compared through 3-grams (Sect. 5.2.1), with more importance given to full words. Correspondences are decided based on heuristic rules and similarities between labels. For example, a subsumption relationship is suggested if one of the labels of a concept in the source ontology is included in the labels of a concept of the target ontology.

TaxoMap also implements a pattern-based approach and *treatments* to help integration engineers refine alignments by considering the specifics of the ontologies to be matched (Sect. 6.1.4). Alignment refinement is designed with two objectives, namely to help engineers in detecting and correcting erroneous correspondences, and to declaratively specify treatments to be performed for processing the resulting alignment, such as ontology merging or restructuring. In particular, by analysing correspondence refinements provided by integration engineers, the idea is to generalise them and to identify groups of correspondences requiring the same refinement treatment. Then, the respective resolving patterns are applied to these groups. Patterns are specified in a mapping refinement pattern language (MRPL) with the help of a graphical user interface, and can be stored and reused in different alignment

⁴<http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>.

refinement tasks. This activity is viewed as an iterative process of alignment refinement and subsequent validation by integration engineers of the resulting updated alignments.

8.1.47 *iMatch (Ben-Gurion University)*

iMatch (Albagli et al. 2012) is a probabilistic interactive ontology matching system based on Markov networks (Sect. 6.4.2). The matching problem is viewed as that of matching labelled graphs. The system takes as input two OWL ontologies, and an optional partial alignment provided by users, and outputs one-to-one alignments. The approach assumes that initial similarity between pairs of nodes is given in advance, e.g., through edit distance (Sect. 5.2.1) or through user input. The system operates in three steps. First, a specific Markov network (Sect. 6.4.2) is built for a given pair of ontologies. The topology of the network is defined based on constraints and rules, such as (i) produce one-to-one alignments, (ii) if two concepts match, then the respective parent nodes frequently match as well. Then, initial match distributions (computed through edit distance) for all possible pairs of concepts are used to initialise evidence potentials of the network. Ultimately, iMatch exploits probabilistic reasoning in the Markov network to compute the final alignment. Besides initial similarities, it also takes into account the ontology structures through a loopy belief (about matching) propagation (Kschischang et al. 2001). Any network node with a posterior probability over a given threshold is considered as a correspondence.

8.2 Instance-Based Systems

Instance-based systems are those taking advantage mostly of instances, i.e., of data expressed with regard to the ontology or data indexed by the ontology.

8.2.1 *T-tree (INRIA Rhône-Alpes)*

T-tree (Euzenat 1994) is an environment for generating taxonomies and classes from objects (instances). It can, in particular, infer dependencies between classes, called bridges, of different ontologies sharing the same set of instances based only on the extension of classes (Sect. 5.4.1). Given a set of source taxonomies called viewpoints, and a destination viewpoint, T-tree returns all the minimal bridges that are satisfied by the available data, i.e., the set of bridges for which the objects in every source class are indeed in the destination class. The algorithm compares the extension (set of instances) of the presumed destination to the intersection of those of the presumed source classes. If there is no inclusion of the latter in the former, the algorithm is re-iterated on all the sets of source classes which contain at least one class

which is a subclass of the tested source classes. If the intersection of the extension of the presumed source classes is included in that of the presumed destination class, a bridge can be established from the latter (and also from any set of subclasses of the source classes) to the former (and also any superclass of the destination class). However, other bridges can also exist on the subclasses of the destination. The algorithm is thus re-iterated on them. It stops when the bridge is trivial, i.e., when the source is empty. Users validate the inferred bridges.

Bridge inference is the search for correlation between two sets of variables. This correlation is particular to a data analysis point of view since it does not need to be valid on the whole set of individuals (the algorithm looks for subsets under which the correlation is valid) and it is based on strict set equality (not similarity). However, even if the bridge inference algorithm has been described with set inclusion, it can be helped by other measurements which will narrow or broaden the search. More generally, the inclusion and emptiness tests can be replaced by tests based on the similarity of two sets of objects (as is usual in data analysis).

The bridge inference algorithm is not dependent on the instance-based interpretation: it depends on the meaning of the operators \subseteq , \cap and $= \emptyset$ -test (which are interpreted as their set-theoretic counterpart in the case of the instance-based algorithms). A second version of the system (with the same properties) uses structural comparison: \subseteq is subtyping, \cap is type intersection and $= \emptyset$ -test is a subtyping test.

8.2.2 CAIMAN (*Technische Universität München and Universität Kaiserslautern*)

CAIMAN (Lacher and Groh 2001) is a system for document exchange, which facilitates retrieval and publishing services among communities of interest. These services are enabled by using semi-automatic ontology matching. The approach focusses on lightweight ontologies, such as web classifications. The main idea behind matching is to calculate a probability between the concepts of two ontologies, by applying machine learning techniques for text classification, e.g., the Rocchio classifier. In particular, based on the documents, a representative feature vector (a word-count, weighted by TFIDF feature vector) is created for each concept in an ontology. Then, the cosine measure is computed for two of those class vectors (Sect. 5.2.1). Finally, with the help of a threshold, the resulting alignment is produced.

8.2.3 FCA-Merge (*University of Karlsruhe*)

FCA-merge uses formal concept analysis techniques to merge two ontologies sharing the same set of instances (Stumme and Mädche 2001). The overall process of merging two ontologies consists of three steps, namely (i) instance extraction,

(ii) concept lattice computation, (iii) interactive generation of the final merged ontology. The approach provides, as a first step, methods for extracting instances of classes from documents. The extraction of instances from text documents circumvents the problem that in many applications there are no individuals which are simultaneously instances of the source ontologies and which could be used as a basis for identifying similar concepts. During the second step, the system uses formal concept analysis techniques (Sect. 5.4.1) in order to compute the concept lattice involving both ontologies. The last step derives the merged ontology from the concept lattice. The produced lattice is explored and transformed by users who further simplify it and generate the taxonomy of an ontology.

The result is a merge rather than an alignment. However, the concepts that are merged can be considered as exactly matched and those which are not can be considered in subsumption relation with their ancestors or siblings.

8.2.4 LSD (*University of Washington*)

Learning Source Descriptions (LSD) is a system for the semi-automatic discovery of one-to-one alignments between the (leaf) elements of source schemas and a mediated (global) schema in data integration (Doan et al. 2001). The main idea behind the approach is to learn from mappings created manually between the mediated schema and some of the source schemas, in order to propose in an automatic manner mappings for the subsequent source schemas. LSD handles XML schemas. A schema is modelled as a tree, whose nodes are XML tag names. The approach works in two phases. During the first (training) phase, useful objects, such as element names and data values, are extracted from the input schemas. Then, from these objects and manually created alignments, the system trains several basic matchers (addressing different features of objects, such as formats, word frequencies, characteristics of value distributions) and a metamatcher. Examples of basic matchers are the WHIRL learner (Sect. 7.5.2) and the naive Bayes learner (Sect. 7.5.1). The metamatcher combines the predictions of basic matchers. It is trained by using a stacked generalisation (learning) technique (Sect. 7.6.1). During the second (matching) phase LSD extracts the necessary objects from the remaining (new) source schemas. Then, by applying the trained basic matchers and the metamatcher on the new objects (the *classification* operation), LSD obtains a prediction list of matching candidates. Finally, by taking into account integrity constraints and applying some thresholds, the final alignment is extracted.

8.2.5 GLUE (*University of Washington*)

GLUE (Doan et al. 2004), a successor of LSD, is a system that employs multiple machine learning techniques to semi-automatically discover one-to-one mappings

(which are sometimes called ‘glue’ for interoperability) between two taxonomies. The idea of the approach is to calculate the *joint distributions* of the classes, instead of committing to a particular definition of similarity. Thus, any particular similarity measure can be computed as a function over the joint distributions. As LSD, GLUE follows a multistrategy learning approach, involving several basic matchers and a metamatcher. The system works in three steps. First, it learns the joint probability distributions of classes of the two taxonomies. In particular, it exploits two basic matchers: the *content learner* (naive Bayes technique; Sect. 7.5.1) and the *name learner* (a variation of the previous one). The metamatcher, in turn, performs a linear combination of the basic matchers. Weights for these matchers are assigned manually. During the second step, the system estimates the similarity between two classes with a user-supplied function of their joint probability distributions. This results in a similarity matrix between terms of two taxonomies. Finally, some domain-dependent, e.g., subsumption, and domain-independent, e.g., if all children of node x match node y , then x also matches y , constraints (heuristics) are applied by using a *relaxation labelling* technique. These are used in order to filter some of the matches out of the similarity matrix and keep only the best ones.

8.2.6 iMAP (University of Illinois and University of Washington)

iMAP (Dhamankar et al. 2004) is a system that semi-automatically discovers one-to-one and, most importantly, complex, e.g., $\langle \text{address}, \text{concat}(\text{city}, \text{street}), = \rangle$, mappings between relational database schemas. The schema matching problem is reformulated as a search in a match space, which is often, very large or even infinite. To perform the search effectively, iMAP uses multiple basic matchers, called searches, e.g., text, numeric, category, unit conversion, each of which addresses a particular subset of the match space. For example, the text searcher considers the concatenation of text attributes, while the numeric searcher considers combining attributes with arithmetic expressions. The system works in three steps (see Fig. 8.5). First, matching candidates are generated by applying basic matchers (the match generator module). Even if a basic matcher, such as the text searcher, addresses only the space of concatenations, this space can still be very large. To this end, the search strategy is controlled by using the *beam search* technique (Russell and Norvig 1995). During the second step, for each target attribute, matching candidates of the source schema are evaluated by exploiting additional types of information, e.g., using the naive Bayes evaluator (Sect. 7.5.1), which would be computationally expensive to use during the first step. These yield additional scores. Then, all the scores are combined into a final one by the similarity estimator module. The result of this step is a similarity matrix between $\langle \text{target attribute}, \text{match candidate} \rangle$ pairs. Finally, by using a set of domain constraints and mappings from the previous match operations (if applicable and available), the similarity matrix is cleaned up, such that only the best matches for target attributes are returned as result (the match selector module). The system is also able to explain the results it produces with the help of the explanation module (see Sect. 11.3 for details).

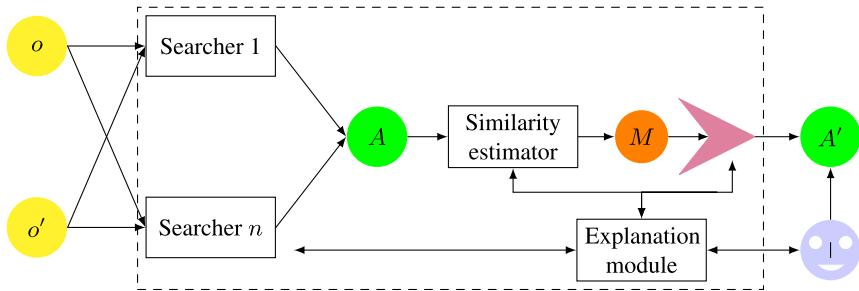


Fig. 8.5 iMAP architecture: several matchers, called searchers, are run in parallel. They provide candidate matches that can be complex. These candidates are further selected by applying the similarity estimator, and then, the final alignment is extracted. Additionally, the explanation module allows users to understand the results and control the process.

8.2.7 Automatch (George Mason University)

Automatch (Berlin and Motro 2002) is a system for automatic discovery of mappings between attributes of database schemas. The approach assumes that several schemas from the domain under consideration have already been manually matched by domain experts. This assumption is a realistic one for a data integration scenario. Then, by using Bayesian learning (Sect. 7.5.1), Automatch acquires probabilistic knowledge from the manually matched schemas, and creates the *attribute dictionary* which accumulates knowledge about each attribute by means of its *possible values* and the *probability estimates* of these values. In order to avoid a rapid growth of the dictionary, the system also uses *statistical feature selection* techniques, such as mutual information, information gain, and likelihood ratio, to learn efficiently, i.e., only from the most informative values, such as 10 % of the available input training data. A new pair of schemas is matched automatically via the precompiled attribute dictionary. The system first matches each attribute of the input schemas against the attribute dictionary, thereby producing individual match scores (a real number). Then, these individual scores are further combined by taking their sum to produce the scores between the attributes of the input schemas. Finally, the scores between the input schemas are again combined by using a *minimum cost maximum flow* graph algorithm and some thresholds in order to find the overall optimal matching between the input schemas with respect to the sum of the individual match scores.

8.2.8 SBI&NB (The Graduate University for Advanced Studies)

SBI (Similarity-Based Integration) is a system for automatic statistical matching among classifications (Ichise et al. 2003, 2004). SBI&NB is the extension of SBI by plugging into the system a naive Bayes classifier (Sect. 7.5.1). The idea of SBI is

to determine correspondences between classes of two classifications by statistically comparing the membership of the documents to these classes. The pairs of similar classes are determined in a top-down fashion by using the κ -statistic method (Fleiss 1973). These pairs are considered to be the final alignment. SBI&NB combines sequentially SBI and naive Bayes learning. The naive Bayes enables hierarchical classification of documents. Thus, the system also takes into account structural information of the input classifications. The exploited classifier is Pachinko Machine naive Bayes from the Rainbow system.²

8.2.9 Kang and Naughton (University of Wisconsin-Madison)

Kang and Naughton proposed a structural instance-based approach for discovering correspondences among attributes of relational schemas with *opaque* column names (Kang and Naughton 2003). Opaque column names are names which are hard to interpret, such as A and B instead of Model and Color. The approach works in two phases. During the first phase, two table instances are taken as input and the corresponding (weighted) dependency graphs are constructed based on *mutual information* and *entropy*. The conditional entropy used describes (with a nonnegative real number) the uncertainty of values in an attribute given knowledge of another attribute (probability distribution). Mutual information, in turn, measures (with a nonnegative real number) the reduction in uncertainty of one attribute due to the knowledge of the other attribute, i.e., the amount of information captured in one attribute about the other. It is zero when the attributes are independent, and increases as the dependency between the two attributes grows. Mutual information is computed over all pairs of attributes in a table. Thus, in dependency graphs, a weight on an edge stands for the mutual information between two adjacent attributes. A weight on a node stands for the entropy of the attribute. During the second phase, matching node pairs are discovered between the dependency graphs by running a graph matching algorithm. The quality of the alignment is assessed by using metrics, such as the Euclidean distance (Sect. 5.2.1). A distance is assigned to each potential correspondence between attributes of two schemas and a one-to-one alignment which is a minimum weighted graph matching (Sect. 7.7.3) is returned.

8.2.10 Dumas (Technische Universität Berlin and Humboldt-Universität zu Berlin)

Dumas (DUPLICATE-based MAching of Schemas) is an approach which identifies one-to-one alignments between attributes by analysing the duplicates in data instances of relational schemas (Bilke and Naumann 2005). Unlike other instance-based approaches which look for similar properties of instances, such as distribution of characters, in columns of schemas under consideration, this approach looks for

similar rows or tuples. The system works in two phases: (i) identify objects within databases with opaque schemas, and (ii) derive correspondences from a set of similar duplicates.

For object identification (Sect. 5.4.2), tuples are viewed as strings and a string comparison metric, such as the cosine measure (Sect. 5.2.1), is used to compare two tuples. Specifically, tuples are tokenised and each token is assigned a weight based on TFIDF (Sect. 5.2.1). In order to avoid complete pairwise comparison of tuples from the two databases, the WHIRL algorithm (Sect. 7.5.2) is used. It performs a focussed search based on those common values that have high TFIDF score. The algorithm ranks tuple pairs according to their similarity and identifies the k most similar tuple pairs.

During the second phase, based on the k duplicate pairs with highest confidence, correspondences between attributes are derived. The intuition is that if two field values are similar, then their respective attributes match. A field-by-field similarity comparison is made for each of the k duplicates, thereby resulting in a similarity matrix. For comparing tuple fields, a variation of a TFIDF-based measure, called soft TFIDF (Cohen et al. 2003a), is used. It considers similar terms as opposed to equal terms. The resulting alignment is extracted from the similarity matrix by finding the maximum weight matching. Finally, if based on the maximum matching, multiple alternative matches are possible, the algorithm iterates back to the first phase to try to improve the result by discovering more duplicates.

8.2.11 Wang and Colleagues (*Hong Kong University of Science and Technology and Microsoft Research Asia*)

Wang and colleagues propose an instance-based solution for discovering one-to-one alignments among web databases (Wang et al. 2004) (see also Sect. 8.1.27). These are query interfaces (HTML forms) and backend databases which dynamically provide information in response to user queries. Authors distinguish between (i) the query interface, which exposes attributes that can be queried in the web database and (ii) the result schema presenting the query results, which exposes attributes that are shown to users. Matching between different query interfaces (inter-site matching) is critical for data integration between web databases. Matching between the interface and result schema of a single web database (intra-site matching), in turn, is useful for automatic data annotation and database content crawling. The approach is based on the following observations (among others):

- The keywords of queries (whose semantics match the semantics of the input element of a query interface) that return results are likely to reappear in attributes of the returned result. For example, such keywords as *Logic* submitted to the input element *title* matches its intended use (while it is not the case with the field *author* which will unlikely produce expected results), and therefore, some results with books about logics will be returned. Moreover, part (*Logic*) of the value *Introduction to logic* of the *title* attribute should reappear in the result schema.

- Based on the work in (He and Chang 2003), the authors assume the existence and availability of a populated global schema, that is a view capturing common attributes of data, for web databases of the same domain of interest.

The approach presents a combined schema model that involves five kinds of schema matching for web databases in the same domain of interest: global-interface, global-result, interface-result, interface-interface, and result-result. The approach works in two phases: (i) query probing and (ii) instance-based matching.

The first phase deals with acquiring data instances from web databases by query probing. It exhaustively sends the attribute values of pre-known instances from a global schema and collects results from the web databases under consideration in a *query occurrence cube*. The cube height stands for the number of attributes in the given global schema. The cube width stands for the number of attributes in the interface schema. The cube depth is the number of attributes in the result schema. Finally, each cell in this cube stores an occurrence count associated with the three dimensions. This cube is further projected onto three *query occurrence matrices*, which represent pairwise relationships between the three schemas, namely global-interface, interface-result and result-global.

During the second phase, the re-occurrences of submitted query keywords in the returned results are analysed. In order to perform intra-site matching, the *mutual information* between pairs of attributes from two schemas is computed (see also Sect. 8.2.9). In order to perform inter-site matching a vector-based similarity is used (Sect. 5.2.1). In particular, each attribute of an individual interface or result schema is viewed as a *document* and each attribute of the global schema is viewed as a *concept*. Each row in the occurrence matrix represents a corresponding document vector. The similarity between attributes from different schemas is computed by using the cosine measure (Sect. 5.2.1) between two vectors. Finally, for both intra-site matching and inter-site matching, the matrix element whose value is the largest both in its row and column represents a final correspondence.

8.2.12 sPLMap (*University of Duisburg-Essen, and ISTI-CNR*)

sPLMap (probabilistic, logic-based mapping between schemas) is a framework which combines logics with probability theory in order to support uncertain schema mapping (Nottelmann and Straccia 2005, 2006). In particular, it is a GLAV-like framework (Lenzerini 2002) where the alignment is defined as uncertain rules in probabilistic Datalog. This allows for supporting imprecise matches, e.g., between author and editor attributes and a more general attribute, such as creator, which is often the case in schemas with different levels of granularity. sPLMap matches only attributes of the same concept (typically documents). The system operates in three main steps. First, it evaluates the quality of all possible individual correspondences on the basis of probability distributions (called interpretation). It selects the set of correspondences that maximises probability on the basis of instance data.

Then, for each correspondence, matchers are used as quality estimators: they provide a measure of the plausibility of the correspondence. SPLMap has been tested with the following matchers: (i) same attribute names (Sect. 5.2.1), (ii) exact tuples (Sect. 5.4), (iii) the k -nearest neighbour classifier (Sect. 7.5.2), and (iv) the naive Bayes classifier (Sect. 7.5.1). The results of these matchers are aggregated by using linear or logistic functions, or their combinations (Sect. 7.4). Coefficients of the normalisation functions are learnt by regression in a system-training phase. Finally, the computed probabilities are transformed in correspondence weights (used as the probability of the corresponding Datalog clause) by using the Bayes theorem.

8.2.13 FSM (Poland National Institute of Telecommunications, Humboldt-Universität zu Berlin, Max Plank Institute for Computer Science)

FSM is an instance-based approach to schema matching (though extensible to schema-level information as well) on data streams or large databases (Jaroszewicz et al. 2008). Exact calculation of similarities between values of attributes is infeasible for data streams. Instead, the approach uses only a small sample of records and guarantees finding an approximate (quasi-optimal) alignment compared to the one obtained by processing the entire data stream. The method can be applied to any combination of similarity metrics that can be estimated with bounded error from a sample. For each attribute in the schemas to be matched, it finds the k -approximately best matching candidates and orders them according to their similarity. Then, the attributes in each schema are sorted approximately according to their likelihood of having a matching candidate in the other schema. An approximately optimal matching problem is formulated, i.e., a stochastic approximation with confidence bounds of the chosen similarity function. Specifically, a weighted sum over Euclidean distance, 2-grams and character proportions are used (Sect. 7.4). Users define two major parameters, namely approximation or desired accuracy (ε) and confidence or error probability (δ), on which the required sample size and execution time depend. Although the method is independent from the amount of data available, a progressive FSM method was also designed. It discovers alignments faster when there are only some similar and many dissimilar elements.

8.2.14 VSBM & GBM (École Centrale Paris)

An automatic extensional approach to ontology matching has been developed in the context of image retrieval applications (James et al. 2010), namely for associating common sense knowledge, such as in WordNet, to specific multimedia concepts, such as LSCOM (Naphade et al. 2006). In particular, two methods were developed: a variable selection-based method (VSBM) and a graph-based method (GBM).

These rely on *visual* and *textual* features of instances, which can be represented as real-valued vectors. Visual matching is used to provide a baseline solution, which is further refined through textual matching. Following the notion of variable selection, VSBM computes similarities between two concepts by comparing ranks assigned to the respective input variables. These similarities are measured through the Spearman correlation measure and n -TF, which measures the intersection of the top n features. In turn, a rank per variable and per concept is computed through pointwise mutual information (co-occurrence counts). Finally, GBM explores relationships between objects, i.e., images and their features. It computes correspondences between concepts through measuring correlations discovered by a random walk in a mixed multimedia graph that encodes both text and visual modalities (Tong et al. 2006).

8.2.15 ProbaMap (*Université de Grenoble*)

ProbaMap is an automatic system for discovering probabilistic subsumption mappings between classes of two populated taxonomies (Tournaire et al. 2011). Among all possible correspondences it determines those that are the most probable based on instance descriptions. For assessing mapping probabilities, a Bayesian estimation is adopted by using the descriptions of instances categorised in each taxonomy as observations of the involved classes. ProbaMap implements a generate-and-test algorithm that minimises the number of calls to the probability estimator for determining those mappings with a probability that is higher than a threshold. The monotony of the probability function is used in order to facilitate pruning of probability estimations for as many mappings as possible. Various classifiers have been used for estimating the probabilities, such as naive Bayes, C4.5 decision tree and SVM as implemented in Weka (Sect. 7.5).

8.3 Mixed, Schema-Based and Instance-Based Systems

The following systems take advantage of both schema-level and instance-level input information if both are available.

8.3.1 SEMINT (*Northwestern University, NEC and The MITRE Corporation*)

SEManic INTegrator (SEMINT) is a tool based on neural networks to assist in identifying attribute correspondences in heterogeneous databases (Li and Clifton 1994, 2000). It supports access to a variety of database systems and uses both schema- and instance-level information to produce rules for matching corresponding attributes

automatically. The approach works as follows. First, it extracts from two databases all the necessary information (features or discriminators) which is potentially available and useful for matching. This includes normalised schema information, e.g., field specifications, such as data types, length, constraints, and statistics about data values, e.g., character patterns, such as ratio of numerical characters, ratio of white spaces, and numerical patterns, such as mean, variance, standard deviation. Second, by using a neural network as a classifier with the self-organising map algorithm (Sect. 7.5.3), it groups the attributes based on similarity of the features for the first database. Then, it uses a back-propagation neural network for learning and recognition. Training is performed on the previously obtained clusters. Finally, using a neural network trained on the first database features and clusters, the system recognises and computes similarities between the categories of attributes from the first database and the features of attributes from the second database, thus, generating a list of match candidates, which are to be inspected and confirmed or discarded by users.

8.3.2 IF-Map (University of Southampton and University of Edinburgh)

IF-Map (Information-Flow-based Map) (Kalfoglou and Schorlemmer 2003a) is an ontology matching system based on the Barwise–Seligman theory of information flow (Barwise and Seligman 1997). The basic principle of IF-Map is to match two local ontologies by looking at how these are related to a common reference ontology. It is assumed that such a reference ontology represents an agreed understanding that facilitates the sharing of knowledge. This means that two local ontologies have significant fragments of them that conform to the reference ontology. It is also assumed that the local ontologies are populated with instances, while the reference ontology does not need to.

Matching works as follows. If the reference ontology can be expressed in each of the local ontologies and instances of the local ontologies can be assigned concepts in the reference ontology (or be mapped to equivalent instances in the reference ontology), then IF-Map uses formal concept analysis (Sect. 5.4.1) between the three ontologies in order to find the Galois lattice from which it is possible to extract an alignment.

When the mappings are not available, IF-Map generates candidate pairs of mappings (called *infomorphism* in information flow theory) and artificial instances. They are generated through the enforcement of constraints that are induced by the definition of the reference ontology and by heuristics based on string-based (Sect. 5.2.1) and structure-based (Sect. 5.3) methods.

IF-Map deals with ontologies expressed in KIF or RDF. The IF-MAP method is declaratively specified in Horn logic and is executed with a Prolog interpreter, so the ontologies are translated into Prolog clauses beforehand. IF-Map produces concept-to-concept and relation-to-relation alignments.

8.3.3 *NOM and QOM (University of Karlsruhe)*

NOM (Naive Ontology Mapping) (Ehrig and Sure 2004) and QOM (Quick Ontology Mapping) (Ehrig and Staab 2004) are components of the FOAM framework (Sect. 10.3.7).

NOM adopts the idea of parallel composition of matchers from COMA (Sect. 8.1.12). Some innovations with respect to COMA are in the set of elementary matchers based on rules, exploiting explicitly codified knowledge in ontologies, such as information about super- and subconcepts, super- and subproperties. As from (Ehrig and Sure 2004), the system supports 17 rules related to those of Table 6.1 (p. 123). For example, a rule states that if superconcepts are the same, the actual concepts are similar to each other. These rules are based on various terminological and structural techniques.

QOM (Quick Ontology Mapping) (Ehrig and Staab 2004) is a variation of the NOM system dedicated to improve the efficiency of the system. The approach is based on the idea that the loss of quality in matching algorithms is marginal (to a standard baseline), however improvement in efficiency can be significant. This fact allows QOM to produce correspondences fast, even for large-size ontologies. QOM is grounded on the matching rules of NOM. However, for the purpose of efficiency the use of some rules, e.g., the rules that traverse the taxonomy, has been restricted. QOM avoids the complete pairwise comparison of trees in favour of an incomplete top-down strategy, thereby focussing only on promising matching candidates.

The similarity measures produced by basic matchers (matching rules) are refined by using a sigmoid function (Sect. 7.7.2), thereby emphasising high individual similarities and de-emphasising low individual similarities. They are then aggregated through weighted average (Sect. 7.4). Finally, with the help of thresholds, the final alignment is produced.

8.3.4 *oMap (CNR Pisa)*

oMap (Straccia and Troncy 2005) is a system for matching OWL ontologies. It is built on top of the Alignment API (Sect. 10.3.6) and has been used for distributed information retrieval in (Straccia and Troncy 2006). *oMap* uses several matchers (called classifiers) that are used for giving a plausibility of a correspondence as a function of an input alignment between two ontologies. The matchers include (i) a classifier based on classic string similarity measure over normalised entity names (Sect. 5.2.1), (ii) a naive Bayes classifier (Sect. 7.5.1) used on instance data, and (iii) a ‘semantic’ matcher which propagates initial weights through the ontology constructors used in the definitions of ontology entities. This last one starts with an input alignment associating plausibility to correspondences between primitive entities and computes the plausibility of a new alignment by propagating these measures through the definitions of the considered entities. The propagation rules depend on

the ontology constructions, e.g., when passing through a conjunction, the plausibility will be minimised. Each matcher has its own threshold and they are ordered among themselves.

There are two ways in which matchers can work: (i) in parallel, in which case their results are aggregated through a weighted average, such that the weights correspond to the credit accorded to each of the classifiers, (ii) in sequence, in which case each matcher only adds new correspondences to the input ontologies. A typical order starts with string similarity, before naive Bayes, and then the ‘semantic’ matcher is used.

8.3.5 Xu and Embley (Brigham Young University)

Xu and Embley proposed a parallel composition approach to discover, in addition to one-to-one alignments, also one-to-many and many-to-many correspondences between graph-like structures, e.g., XML schemas, classifications (Xu and Embley 2003; Embley et al. 2004). Schema matching is performed by a combination (an average function) of multiple matchers and with the help of external knowledge resources, such as domain ontologies. The basic element-level matchers used in the approach include *name matcher* and *value-characteristic matcher*. The name matcher, besides string comparisons (Sect. 5.2.1), also performs some linguistic normalisation, such as stemming and removing stop words (Sect. 5.2.2). It also detects synonyms among node names with the help of WordNet (Sect. 5.2.2). Matching rules are obtained via a C4.5 decision tree generator (Sect. 7.5.5) that has been trained over WordNet by using several hundreds synonym names found in the available databases from a domain of interest. The value-characteristic matcher determines where two values of schema elements share similar value characteristics, such as means or variances for numerical data. Similar to the name matcher, matching rules are obtained by training the C4.5 decision tree generator over value characteristics of the available databases from a domain of interest. Structure-level matchers are used to suggest new correspondences as well as to confirm correspondences identified by element-level matchers, for example, by considering similarities between the neighbour elements computed by element-level matchers. Another structural matcher tries to match both schemas to an external domain ontology in order to decide if they match.

8.3.6 Wise-Integrator (SUNY at Binghamton, University of Illinois at Chicago and University of Louisiana at Lafayette)

Wise-Integrator is a tool that performs automatic integration of Web Interfaces of Search Engines (He et al. 2004, 2005). It provides a unified interface to e-commerce search engines of the same domain of interest, such as books and music. Therefore,

users can pose queries by using this interface and the search mediator sends the translated subqueries to each site involved in handling this query and then the results of these sites are reconciled and presented to users. Wise-Integrator consists of two main subsystems: (i) an interface schema extractor, and (ii) an interface schema integrator. The first component, given a set of HTML pages with query interfaces, identifies logical attributes and derives some meta-information about them, e.g., data type, thereby building an interface schema out of them. For example, the system can derive (guess) that the field Publication Date, is likely to be of date data type. The second component discovers matching attributes among multiple query interfaces and then merges them, thereby resulting in global attributes. These are used, in turn, to produce a unified search interface.

Attribute matching in Wise-Integrator is based on two types of matches: *positive* and *predictive*. Positive matches are based on the following matching methods: exact name match, look-up for synonymy, hypernymy and meronymy in WordNet (Sect. 5.2.2), and value-based matchers. When one of the positive matches occurs, the corresponding attributes are considered as matched. Predictive matches are based on the following matching methods: approximate name match, e.g., edit distance (Sect. 5.2.1), data type compatibility (Sect. 5.3.2), value pattern matcher (Sect. 5.4.3). Predictive matches have to be strong enough (which is decided based on a threshold) in order to indicate that the attributes under consideration match.

Positive and predictive matches are used in two successive clustering steps: *positive match-based clustering* and *predictive match-based clustering*. In the first step, all the interfaces are taken as input and attributes are grouped into clusters based on the positive matches between attributes. Clustering is done by following pre-defined rules that govern the order of execution of underlying matchers and how to make groupings based on results of these matchers. For example, the first results of exact name matches are considered before results of value-based and WordNet-based matchers. Finally, for each cluster a representative attribute name (RAN) is determined. For example, for the cluster with attribute names {Format, Binding type, Format} the RAN is Format. During the second step, all local interfaces are reconsidered again. Clustering is performed following some pre-defined rules that employ previously determined RANs and a simple weighting scheme over the results of predictive matching methods. When all potentially matching attributes are clustered together, the global attribute for each group of such attributes is generated.

8.3.7 *IceQ (University of Illinois at Urbana-Champaign, University of Illinois at Chicago, SUNY at Binghamton)*

IceQ is an interactive system for matching interfaces on the deep web (Wu et al. 2004). Web interfaces are modelled as trees, rather than flat lists of attributes. IceQ discovers 1:1 and 1: m correspondences. It uses clustering based on label and instance (domain) similarity to group attributes that match. All the attributes are considered holistically at once between all the web interfaces. Similarity between labels

is computed through cosine function over vectors of normalised words (Sect. 5.2.1) extracted from attribute labels. In turn, domain similarity is based on the derived types and the actual values of the domains. Types are usually not specified in the interface, so these are derived based on value patterns. Type examples are money, area or date. Character values are compared through the cosine similarity for every pair of values. Numeric values are compared as a percentage of the overlapping range values in the domains. Label and domain similarities are further aggregated as a linear combination (Sect. 7.4.1) with more importance (heavier weight) given to the label similarity. IceQ also employs active learning to determine a cluster stopping threshold to apply, possible synonyms and homonyms. For this purpose, within the matching process, the system selectively poses some yes/no questions to users and shows both labels and instances of the attributes under consideration.

The system has been enhanced with the WebIQ component, which automatically acquires necessary instances for interface attributes in order to improve matching accuracy (Wu et al. 2006). This is achieved, for example, through borrowing data instances from other attributes, or through discovering them on the surface web and finally validating them via the deep web.

8.3.8 OLA (*INRIA Rhône-Alpes and Université de Montréal*)

OLA (OWL Lite Aligner) (Euzenat and Valtchev 2004) is an ontology matching system designed with the idea of balancing the contribution of each of the components that compose an ontology, e.g., classes, constraints, data instances. OLA handles ontologies in OWL. It first compiles the input ontologies into graph structures, unveiling all relationships between entities. These graph structures produce the constraints for expressing a similarity between the elements of the ontologies. The similarity between nodes of the graphs follows two principles: (i) it depends on the category of node considered, e.g., class, property, and (ii) it takes into account all the features of this category, e.g., superclasses, properties, as presented in Table 6.1.

The distance between nodes in the graph is expressed as a system of equations based on string-based (Sect. 5.2.1), language-based (Sect. 5.2.2) and structure-based (Sect. 5.3) similarities (as well as taking instances into account whenever necessary). These distances are almost linearly aggregated (they are linearly aggregated modulo local matches of entities). For computing these distances, the algorithm starts with base distance measures computed from labels and concrete data types. Then, it iterates a fixed point algorithm until no improvement is produced. From that solution, an alignment is generated which satisfies some additional criterion on the obtained alignment and the distance between matched entities. The algorithm is described in more detail in Sect. 6.2.2. The OLA architecture is typically the one displayed in Fig. 6.4 (p. 131).

8.3.9 Falcon-AO (China Southeast University)

Falcon is an automatic divide-and-conquer approach to ontology matching (Hu et al. 2008). It handles ontologies in RDFS and OWL. It has been designed with the goal of dealing with large ontologies (of thousands of entities). The approach operates in three phases: (i) partitioning ontologies, (ii) matching blocks, and (iii) discovering alignments. The first phase starts with a structure-based partitioning to separate entities (classes and properties) of each ontology into a set of small clusters. Partitioning is based on structural proximities between classes and properties, e.g., how close the classes are in the hierarchies of `rdfs:subClassOf` relations, and on an extension of the ROCK agglomerative clustering algorithm (Sect. 7.1.1). Then it constructs blocks out of these clusters. In the second phase the blocks from distinct ontologies are matched based on anchors (pairs of entities matched in advance), i.e., the more anchors between two blocks, the more similar the blocks. In turn, the anchors are discovered by matching entities with the help of the SMOA string comparison technique (Sect. 5.2.1). The block pairs with high similarity are selected based on a cutoff threshold. Each block is just a small fragment of an ontology. Finally, the third phase combines two matchers between the matched block pairs via sequential composition (see Fig. 8.6):

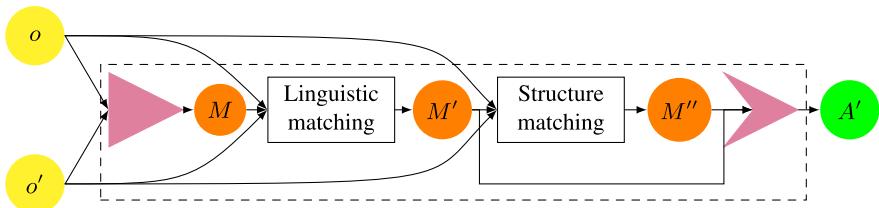


Fig. 8.6 Falcon-AO architecture: it is a sequential composition of two components, but if the output of the linguistic matcher is considered of sufficient quality, then no structure matching is performed.

LMO is a linguistic matcher. It associates with each ontology entity a bag of words which is built from the entity label, the entity annotations as well as the labels of connected entities (called virtual documents). The similarity between entities is based on TFIDF (Sect. 5.2.1) (Qu et al. 2006).

GMO is a bipartite graph matcher (Hu et al. 2005). It starts by considering the RDF representation of the ontologies as a bipartite graph which is represented by its adjacency matrix (A and A'). The distance between the ontologies is represented by a distance matrix (X) and the distance (or update) equations between two entities are simply a linear combination of all entities they are adjacent to, i.e., $X^{t+1} = AX^t A'^T + A^T X^t A'$. This process can be bootstrapped with an initial distance matrix. However, the real process is more complex than described here because it distinguishes between external and internal entities as well as between classes, relations and instances.

First LMO is used for assessing the similarity between ontology entities on the basis of their name and text annotations. If the result has a high confidence, then it is directly returned for extracting an alignment. Otherwise, the result is used as input for the GMO matcher which tries to find an alignment on the basis of the relationships between entities (Jian et al. 2005). Ultimately, the output alignment is extracted through a greedy selection (Sect. 7.7).

8.3.10 RiMOM (*Tsinghua University*)

The RiMOM (Risk Minimisation-based Ontology Mapping) approach, being inspired by Bayesian decision theory, formalises ontology matching as a decision making problem (Tang et al. 2006). Given two ontologies, it aims at an optimal and automatic discovery of alignments which may be complex (such as including concatenation operators). The approach first searches for concept-to-concept correspondences and then for property-to-property correspondences. The RiMOM matching process is organised into the following phases (Li et al. 2006):

1. Select matchers to use. This task can be performed either automatically or manually. The basic idea of automatic strategy selection is that if two ontologies have a high label similarity factor, then RiMOM will rely more on linguistic-based strategies; while if the two ontologies have a high structure similarity factor, RiMOM will exploit similarity-propagation based strategies on them.
2. Execute multiple independent matchers, given the input ontologies and, optionally, user input. Examples of matchers include linguistic normalisation of labels, such as tokenisation, expansion of abbreviations and acronyms (Sect. 5.2.2) based on GATE (Sect. 5.2.2), edit distance, matchers that look for label similarity based on WordNet (Sect. 5.2.2), k -nearest neighbours statistical learning, naive Bayes matcher (Sect. 7.5.1), as well as some other heuristics for data type similarity and taxonomic structure similarity. This results in a similarity in [0 1] for each pair of entities from the two ontologies (see also Sect. 8.1.12).
3. Combine the results by aggregating the values produced during the previous step into a single value. This is performed by using a linear interpolation.
4. Propagate similarity. If the two ontologies have a high structure similarity factor, RiMOM employs an algorithm called similarity propagation to refine the found alignments and to find new alignments that cannot be discovered using the other strategies. Similarity propagation makes use of structure information.
5. Extract alignment for a pair of ontologies based on thresholds (Sect. 7.7.1) and some refinement heuristics to eliminate unreasonable correspondences, e.g., use concept-to-concept correspondences to refine property-to-property correspondences.
6. Iterate the above described process by taking the output of one iteration as input into the next iteration until no new correspondence is produced. At each iteration, users can select matchers, and approve and discard correspondences from the returned alignment

RiMOM offers three possible structural propagation strategies: concept-to-concept propagation strategy (CCP), property-to-property propagation strategy (PPP), and concept-to-property propagation strategy (CPP). For choosing between them, RiMOM uses heuristic rules. For example, if the structure similarity factor is lower than some threshold then RiMOM does not use the CCP and PPP strategies, only CPP is used.

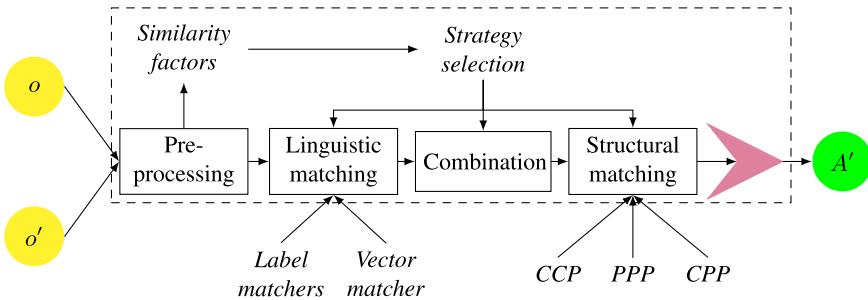


Fig. 8.7 RiMOM architecture.

A new version of RiMOM (Li et al. 2009) is a dynamic metastrategy ontology matching framework (see Fig. 8.7). It quantitatively estimates the similarity characteristics, called similarity factors, for each matching task. Two basic matching methods are employed: (i) linguistic matching (edit distance over entity labels, vector distance among comments and instances of entities) and (ii) structural matching, a variation of Similarity flooding (Sect. 6.2.1) implemented as the three similarity propagation strategies mentioned above, namely CCP, PPP, and CPP. In turn, the strategy selection uses label and structure similarity factors, obtained by preprocessing the ontologies to be matched, in order to determine what information should be employed in the matching process. Specifically, the strategy selection dynamically regulates the concrete feature selection for linguistic matching, the combination of weights for similarity combination, and the choice of the concrete similarity propagation strategy. After similarity propagation, the matching process concludes with alignment refinement and extraction of the final result.

8.3.11 Corpus-Based Matching (University of Washington, Microsoft Research and University of Illinois)

Madhavan and colleagues (Madhavan et al. 2005) proposed an approach to schema matching which, besides input information available from the schemas under consideration, also exploits some domain-specific knowledge via an external corpus of schemas and mappings. The approach is inspired from the use of corpus in information retrieval, where similarity between queries and concepts is determined based on

analysing large corpora of text. In schema matching, such a corpus can be initialised with a small number of schemas obtained, for example, by using available standard schemas in the domain of interest, and may evolve in time with new matching tasks.

Since the corpus is intended to have different representations of each concept in the domain, it should facilitate learning these variations in the elements and their properties. The corpus is exploited in two ways. First, to obtain additional evidence about each element being matched by including evidence from similar elements in the corpus. Second, in the corpus, similar elements are clustered and some statistics for clusters are computed, such as neighbourhood and ordering of elements. These statistics are ultimately used to build constraints that facilitate selection of the correspondences in the resulting alignment.

The approach handles web forms and relational schemas and focusses on one-to-one alignments. It works in two phases. Firstly, schemas under consideration are matched against the corpus, thereby augmenting these with possible variations of their elements based on knowledge available from the corpus. Secondly, augmented schemas are matched against each other. In both cases, the same set of matchers is applied. In particular, basic matchers, called learners, include (i) a *name learner*, (ii) a *text learner*, (iii) a *data instance learner*, and (iv) a *context learner*. These matchers mostly follow the ideas of techniques used in LSD (Sect. 8.2.4) and Cupid (Sect. 8.1.11). For example, the *name learner* exploits names of elements. It applies tokenisation and n -grams (Sect. 5.2.1) to the names in order to create training examples. The matcher itself is a text classifier, such as naive Bayes (Sect. 7.5.1). In addition, the name learner, uses edit distance (Sect. 5.2.1), in order to determine similarity between element names. The *data instance learner* determines whether the values of instances share common patterns, same words, etc. A matcher, called *metalearner*, combines the results produced by basic matchers. It uses *logistic regression* with the help of the stacking technique (Sect. 7.6.1) in order to learn its parameters. Finally, candidate correspondences are filtered by using constraints based on the statistics obtained from the corpus, in order to produce the final alignment.

8.3.12 iMapper (*Norwegian University of Science and Technology*)

iMapper is a semi-automatic ontology matching system (Su and Gulla 2006). It handles ontologies in RML (Referent Ontology Language), an extended entity–relationship language. The matching process is organised in two phases: (i) ontology enrichment and (ii) matching. The enrichment phase uses ontology instances or documents associated with its concepts. In turn, these are exploited to construct feature vectors (through TFIDF; Sect. 5.2.1) for each concept in the ontologies to be matched. A linguistic classifier, called classification and search, is used to assign documents to the ontologies (Brasethvik and Gulla 2001). User intervention is expected to verify the correctness of the assignments made by the classifier, and adjust them, if necessary. The matching phase is organised as follows. First, similarities between concepts are computed through the cosine measure between the respective

feature vectors. Then, the WordNet path length (Sect. 5.2.2) is used to enhance and re-rank the list of previously computed similarities. Based on the similarities between concepts, the respective similarities between relations and concept clusters are computed. For example, similarity between relations is computed as the arithmetic mean of the similarities of the respective domain and range concepts. The system provides a graphical user interface, so that users can validate or edit correspondences. Concept clusters help users focus on similar areas when inspecting the correspondences. Finally, the exporter module translates the alignments into a desired format.

8.3.13 SAMBO (*Linköpings University*)

SAMBO is a system for matching and merging biomedical ontologies (Lambrix and Tan 2006). It handles ontologies in OWL and outputs 1:1 alignments between concepts and relations. The system uses various similarity-based matchers, including:

- *terminological*: n -gram, edit distance, comparison of the lists of words of which the terms are composed. The results of these matchers are combined via a weighted sum with pre-defined weights;
- *structural*, through an iterative algorithm that checks if two concepts occur in similar positions with respect to *is-a* or *part-of* hierarchies relative to already matched concepts, with the intuition that the concepts under consideration are likely to be similar as well;
- *context-based*, using (i) a relationship between the matched entities in UMLS (Sect. 2.1.2) and (ii) a corpus of knowledge collected from the published literature exploited through a naive Bayes classifier (Sect. 7.5.1).

The results produced by these matchers are combined based on user-defined weights. Then, filtering based on thresholds is applied to come up with an alignment suggestion, which is further displayed to users for feedback (approval, rejection or modification). Once matching has been accomplished, the system can merge the matched ontologies, compute the consequences, or check the newly created ontology for consistency. SAMBO was subsequently extended into a toolkit for evaluating ontology matching strategies, called KitAMO (Lambrix and Tan 2007).

8.3.14 AROMA (*University of Nantes, INRIA*)

AROMA is an Association Rule Ontology Matching Approach (David et al. 2006, 2007). It is based on the association paradigm and a (statistical) implication intensity measure. It is able to discover equivalences and subsumptions between entities of web directories and OWL ontologies. The association rules are *if antecedent, then consequent* propositions, which can be interpreted as subsumptions. The underlying

intuition is as follows: an entity e is equivalent or more specific than an entity e' if its vocabulary, made of terms and instances, comprising also those of its descendants, tends to be included in the vocabulary of e' . The matching process is organised in three steps:

- Extraction of a set of the relevant stemmed terms for each concept and property.
- Discovery of binary association rules between entities, by evaluating statistically quasi-implications, i.e., allowing for some counter-examples, based on the respective sets of relevant terms. Selection of significant rules is grounded on: (i) an assessment of the implication quality through the implication intensity measure, (ii) an assessment of the generativity of the rule and reducing its redundancy.
- Cleaning, e.g., by removing redundant correspondences, and enhancing, e.g., by using string equality and the Jaro–Winkler string similarity (Sect. 5.2.1), of the final alignment.

8.3.15 ILIADS (*University of Maryland, University of Toronto*)

ILIADS (Integrated Learning in Alignment of Data and Schema) is a system for integrating ontologies that couples statistical and logical inference (Udrea et al. 2007). In particular, it combines a clustering algorithm with incremental rule-based logical inference. ILIADS performs both schema and instance matching in an interleaved manner. It provides schema-level and instance-level correspondences as well as an integrated consistent ontology.

The approach finds candidate clusters of similar entities. Then, based on rules it individuates an equivalence or subsumption correspondences between the sets of entities. For example, if a set of instances of one concept is almost a subset of a set of instances of another concept (what is quantified through a threshold), then a subsumption relation should hold (see also Sect. 5.4). Hierarchical agglomerative clustering is applied separately to classes, properties and instances. Similarity between clusters is based on a linear combination of lexical (Jaro–Winkler and WordNet), structural (Jaccard similarity on neighbour subclasses or subproperties) and extensional (Jaccard similarity on sets on instances) similarities between entities. Similarity values are propagated incrementally based on clustering, which considers relations among sets of equivalent entities. Rule-based logical inference is used to evaluate whether two clusters must be merged by checking for inconsistencies. It also provides an estimate of the logical consequences of updating the similarities among the remaining clusters.

8.3.16 SeMap (*Georgia Tech, University of British Columbia*)

SeMap is a system that constructs generic mappings across data models (Wang and Pottinger 2008). It discovers 1:1, 1: n , n :1 relationships of four types, namely, *equivalence*, *has-a*, *is-a* and *associates*. Its key idea is to search for a globally optimal

match assignment from a pool of possible assignments, solving conflicts among the selected matches by involving users and identifying a generic relationship between these matches. Specifically, the matching process is organised as follows:

- *Schema matching* identifies candidate correspondences. A linear combination is performed from scores computed by a label matcher (that includes tokenisation, stop word elimination, etc.), a data type matcher, a WordNet sense matcher (Sect. 5.2.2), and a data instance matcher that uses data formats and distributions (Sect. 5.4.3). These combined similarities are employed as input to a structure matcher, implementing Similarity flooding (Sect. 6.2.1), which provides the final similarity score for each pair of elements.
- *Match selection*, given previously computed matches, searches a global optimal match assignment that satisfies a set of domain constraints. Hence, match selection is reduced to a constrained optimisation problem (Sect. 7.7.3). It results in two sets of correspondences that stand for the source (Map_s) and the target (Map_t) perspectives. Users are involved to disambiguate several cases or to provide correct match assignments. In particular, elements are ranked based on their potential information value and user feedback is asked only for those with highest (mutual information) values. Then, the rest of the assignment is updated according to this feedback.
- *Mapping assembling* combines available mappings (Map_s and Map_t) into a final one. Relationship types for the selected matches are identified based on rules and matchers used, e.g., similar names suggest the equivalence relation; if instances of two elements intersect, this suggests the associate relation.

8.3.17 ASMOV (INFOTECH Soft, Inc., University of Miami)

ASMOV (Automatic Semantic Matching of Ontologies with Verification) is an automatic approach for ontology matching that targets information integration for bioinformatics (Jean-Mary et al. 2009). The approach can be summarised in two steps: (i) similarity calculation, and (ii) semantic verification (see Fig. 8.8). It takes as input two OWL ontologies and an optional input alignment and returns as output an $n:m$ alignment between ontology entities (classes and properties). In the first step, it uses lexical (string equality, a variation of the Levenshtein distance), structural (weighted sum of the domain and range similarities) and extensional matchers to iteratively compute similarity measures between two ontologies, which are then aggregated into a single one with a weighted average. It also uses several sources of general and domain-specific background knowledge, such as WordNet and UMLS, to provide more evidence for similarity computation (see also Sect. 7.3). Then, it derives an alignment and checks it for inconsistency. Consistency checking is (anti)pattern-based (Sect. 7.8.2), i.e., that instead of using a complete solver, the system recognises sets of correspondences that are proved to lead to an inconsistency. The semantic verification process examines five types of patterns, e.g., disjoint-subsumption

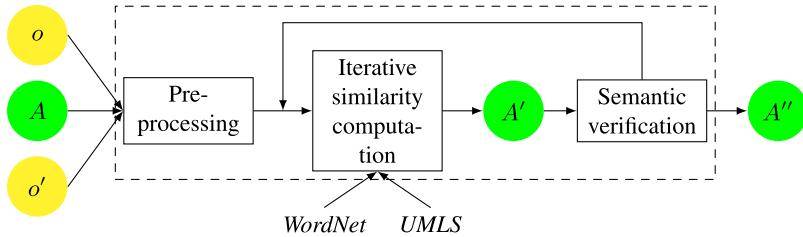


Fig. 8.8 ASMOV architecture.

contradiction, or subsumption incompleteness. This matching process is repeated with the obtained alignment as input until no new correspondence is found.

8.3.18 HAMSTER (University of Michigan, Microsoft Research)

HAMSTER (Human Assisted Mapping of Schema and Taxonomies to Enhance Relevance) is a system for unsupervised matching of schemas and taxonomies from a large number of data sources into the schema and taxonomy of a data warehouse (Nandi and Bernstein 2009). It is aimed at using structured data sources to enhance the results of keyword-based web search. In such a setting, data, such as shopping lists, coming from third-party providers have to be integrated into a data warehouse that is indexed by the search engine for keyword queries.

The system handles schemas and taxonomies in XML. Matching between these is viewed as node matching of the underlying tree structures. The approach, besides schema-based methods, e.g., lexical similarities of labels, and instance-based methods, e.g., naive Bayes over content of entities (Sect. 7.5.1), uses an additional source of information, such as query logs extracted from a search engine. Specifically, click-through data, namely search results that users clicked on, is used. The key intuition is that if two items in a database are similar, then they should be searched by using similar queries. Technically, two schema or taxonomy entities are matched in a pairwise fashion if the normalised frequency distribution of keyword queries that cause click-throughs on their instances are similar. The similarity alternatives investigated are the Jaccard similarity on words (Sect. 5.4.1) and exact string comparison (Sect. 5.2.1). Tunable thresholds are used to select candidate correspondences. Finally, schema-based, instance-based and query distribution methods are combined in a consensus algorithm by summing the normalised scores for each candidate correspondence to estimate the impact of various features on matching quality.

8.3.19 Smart Matcher (Vienna University of Technology)

Smart Matcher is a system that combines matching and transformation through a mapping layer in order to bridge alignments, mappings and executable transforma-

tion code (Wimmer et al. 2009). It finds mappings between two schemas sharing a common set of instances describing the same real-world objects. It uses a dedicated mapping language, whose mappings can be executed directly as transformations. The results of the transformations can be compared, thus contributing to their evaluation. The integration process is organised as follows:

- *Instance generation*: Sample instances for the source and target schemas that represent the same real-world objects are developed. These serve verification purposes, namely for calculating differences between manually created instances which are transformed through mappings. This is the only step requiring user input. It is also conjectured that such instances can be used for increasing the quality of correspondences, similarly to QuickMig (Sect. 8.1.13).
- *Seed matching*: Initial alignments are provided with the help of COMA++ (Sect. 8.1.12), FOAM (Sect. 10.3.7) or any external matcher able to produce alignments in the Alignment format (Sect. 10.1.5).
- *Alignment interpretation*: Previously produced alignments are transformed by *mapping engine* into an executable mapping model.
- *Instance transformation* is performed by the *transformation engine* which executes the current alignment as a transformation.
- *Difference calculation* is performed through a *fitness function* that heuristically compares source and target instances in terms of contained objects, values and links. Then, the presumably missing or wrong objects, values or links are propagated back to the mapping engine.
- *Alignment adaptation*: The current alignment is adapted by the mapping engine, by looking for a suitable adaptation through various strategies. For example, by verifying locally that a mapping can be appropriately applied for a given set of schema elements and globally by choosing the sequence of transformations to apply, such as first those between classes and then those between attributes. The process goes back to *Alignment interpretation*.

The process terminates when no differences are produced or a particular threshold is reached.

8.3.20 GEM/Optima/Optima+ (University of Georgia, Wright State University)

GEM (Doshi et al. 2009) is an iterative approach to ontology matching based on the Generalised Expectation-Maximisation method (Sect. 6.3.1). It inputs ontologies in RDF and OWL and outputs many-to-one concept alignments. Ontologies are represented as directed labelled graphs. The matching problem is viewed as a maximum likelihood problem, i.e., finding the most likely alignment between two ontologies, where likelihood is computed iteratively through the expectation-maximisation (EM) method (Dempster et al. 1977).

Essentially, the EM technique computes expected values of the hidden variables (correspondences) using the observed instances and a previous estimate of the model (match matrix). The approach uses a generalised version of EM, which relaxes the maximisation objective and requires the selection of a mixture model that is better than the previous one. Random sampling is used to come up with a representative set of mixture models, out of which the candidate is selected.

The approach was extended within the Optima system (Thayavaram and Doshi 2011). Specifically, the system computes many-to-many alignments and the Smith–Waterman similarity combined through a 3D sigmoid with WordNet-based measures, such as Lin and gloss-based cosine similarity (Sect. 5.2.2). The Optima+ system (Thayavaram and Doshi 2012) is a new implementation of Optima targeting efficiency. In particular, it employs block coordinate descent to improve the convergence of the iterative process as well as ontology partitioning (Sect. 7.1.1) to handle more efficiently large ontologies.

8.3.21 CSR (*University of the Aegean, Institution of Informatics and Telecommunications*)

CSR (Classification-based learning of Subsumption Relations) is a generic method for automatic ontology matching between concepts based on supervised machine learning (Spiliopoulos et al. 2010). It specifically focusses on discovering subsumption correspondences. CSR assesses a hypothesis concerning a subsumption relation holding between concept pairs through a binary classification task. This is done based on the best fit to the training examples and generalising beyond them. For the training of classifiers, the approach uses traditional TFIDF strategies on statistically generated examples (latent classification features) from the input ontologies. CSR investigated the use of various classifiers, such as naive Bayes, KNN, SVM, C4.5 (Sect. 7.5), and data set balancing methods, ensuring that the training examples of all classes are equal in number, such as random over-sampling, under-sampling and synthetic sampling. The implementation of the classifiers are those of the Weka and the libSVM toolkits. Given a pair of ontologies in OWL-DL, CSR processes as follows:

- *enhance ontology hierarchies*, infer, with the help of reasoning, all subsumption relations separately in each ontology;
- *generate features for the classifier* based on properties of concepts, TFIDF of the extracted words from labels, comments or instances, latent features, etc.;
- *generate training examples*, i.e., pairs of concepts, based on rules, such as, include all concept pairs from both input ontologies that are related through subsumption, and involves also training set balancing;
- *train classifier* with the training data set;
- *generate testing pairs* or match the input ontologies by classifying their concept pairs through the trained classifier.

During the matching step, the search space is pruned in order to exclude pairs of concepts for which a subsumption relation cannot presumably hold, e.g., due to the computed relations.

8.3.22 Prior+ (SAP Labs, Yahoo!, University of Pittsburgh)

Prior+ is a system implementing a generic and adaptive ontology matching approach (Mao et al. 2010). The system consists of the following main modules: (i) information retrieval-based similarity generator, (ii) adaptive similarity filter and weighted similarity aggregator, (iii) constraint satisfaction solver based on a neural network. The matching process is organised as follows. First, linguistic and structural similarities between ontologies are measured in a vector space model using information retrieval methods. Specifically, three similarity measures are employed, after entity label preprocessing:

- name similarity, based on the Levenshtein distance (Sect. 5.2.1),
- profile similarity, generated for each ontology entity, and based on entity features, e.g., IDs, labels, and comments, represented as TFIDF vectors used with cosine similarity (Sect. 5.2.1);
- structural similarity, which is computed for classes only and is based on such features as the number of direct properties of the class, the number of its instances, and the number of its children.

Then, these similarities are aggregated using an adaptive weight assignment method based on the estimated *harmonies* of these similarities (Sect. 7.4.1). Finally, an interactive activation and competition neural network (Sect. 7.5.3) is selectively invoked to satisfy ontology constraints, e.g., cardinality of a property, and to improve the overall accuracy from the previously aggregated results. The harmony parameter is used as an indication as to whether the neural network should be activated or not. The resulting alignments are extracted through a naive descendant extraction algorithm.

8.3.23 YAM & YAM++ (University of Montpellier, University of Toronto)

YAM (not Yet Another Matcher) is a schema matcher factory (Duchateau et al. 2009). It aims at generating a dedicated matcher for a matching task under consideration. The approach is based on machine learning and matchers are viewed as classifiers (Sect. 7.5). The background intuition is that matching algorithms, which combine various similarities, provide diverse results on different matching tasks. YAM, in order to allow for such diversity, learns over a large set of different matchers and different similarity measures. Then, within this search space, using a small

amount of training data, YAM selects and tunes a dedicated matcher for a given matching task (Sect. 7.6), e.g., through weights and thresholds. The approach is extendable and it uses a library of classifiers and of similarity measures. Specifically, it includes 20 classifiers from the Weka library, e.g., decision trees (J48, NBTree), aggregation functions (simple logistic), lazy classifiers (IBk, K*), rule-based (NNge, Jrip) and Bayes Networks (Sect. 7.5). It also includes 30 similarity measures, e.g., Levenshtein, and Jaro–Winkler, from the second string package (Sect. 5.2.1). The process is organised in two steps: (i) training of matchers, and (ii) dedicated matcher selection. YAM uses correspondences stored in an internal repository to produce a matcher trained with best average results for each classifier. A dedicated matcher for a matching task under consideration is selected based on the highest F-measure obtained on the given training data cross-validated with the correspondences in the repository.

The system was extended under the YAM++ name (Ngo et al. 2011; Ngo and Bel-lahsene 2012). It handles multilingual ontologies. More similarity measures were included: such as Smith–Waterman, from the SimMetrics package (Sect. 5.2.1), string-based methods (Sect. 5.2.1), such as prefix, suffix, SMOA, and language-based methods (Sect. 5.2.2), such as Lin and Wu–Palmer metrics over WordNet. YAM++ also introduced profiles capturing relational and extensional information. Specifically, the *IndividualProfile* of an entity is a string concatenation of its identifier, label and comments. The *StructureProfile* of an entity is a union of individual-Profiles of its neighbour entities, e.g., of its subclasses and all restricted properties. The *ExtensionalProfile* of an entity is a string concatenation of instances belonging to this class or its descendants. Similarities between these profiles are computed through information retrieval techniques. The system also included a variation of the Similarity flooding algorithm (Sect. 6.2.1), several alignment extraction methods, such as the Hungarian method (Sect. 7.7.3), as well as semantic verification of inconsistent correspondences by exploiting conflict patterns and the ALCOMO diagnosis library (Sect. 7.8.2). Finally, the system is equipped with a graphical user interface, so that users can provide (optional) input by specifying preferences between precision and recall or by providing partial input alignment or by modifying correspondences.

8.3.24 MoTo (*University of Bari*)

MoTo is an automatic system for Mapping Ontology To Ontology (Esposito et al. 2010). It handles ontologies in OWL and computes equivalences between concepts. The matching process is organised in two phases. In the first phase various basic matchers, implemented as learners, are used to produce single predictions. These are: k -nearest neighbour classifier, neural network (Sect. 7.5.3), and two Bayesian classifiers for content and names (Sect. 7.5.1). The predictions produced by these matchers are combined through the stacked generalisation metalearner (Sect. 7.6.1). The alignments resulting from this phase are classified into certain, uncertain and

those to be discarded. This is done with the help of two thresholds. The second phase is meant to resolve the uncertain alignments through validation. Two types of validation were employed, namely, taxonomic validation and structural validation. These use taxonomic and other relations between concepts, respectively (Sect. 6.1). For example, taxonomic validation compares two concepts by looking at their positions in the respective subsumption hierarchies (Sect. 6.1.1). Finally, aggregation of the individual validatory predictions is performed through ordered weighted average (Sect. 7.4.1) and linguistic quantifiers are used to determine weights (Sect. 7.4.1). This leads to the final decision concerning uncertain alignments.

8.3.25 CODI (*University of Mannheim*)

CODI (Combinatorial Optimization for Data Integration) is a probabilistic-logical ontology matching system (Niepert et al. 2010; Huber et al. 2011). It provides a declarative framework for matching classes, properties and individuals. It is based on Markov logic networks (Sect. 6.4.2). The matching problem is reduced to a maximum a posteriori inference in the Markov logic network, which is in turn solved by using integer linear programming. Given a set of constants, e.g., classes, a set of formulae, e.g., axioms holding between classes, and confidence values for correspondences, a Markov logic network defines a probability distribution over possible correspondences. An input a priori confidence value for each matching hypothesis is computed through a combination of several string-based methods from the Simmetries package (Sect. 5.2.1), such as Levenshtein, Jaro–Winkler and Jaccard. Several combination strategies are supported, such as average, maximum, specific predefined weights assigned to each measure. Pellet (Sirin et al. 2007) is used to create the ground Markov logic network formulation. TheBeast (Riedel 2008) is used to convert the Markov logic network to the corresponding integer linear programming instance, and a mixed integer programming solver is used to ultimately resolve it. The approach handles various constraints, such as cardinality, e.g., returning 1:1 alignments, incoherence mitigation within the matching process, and stability, e.g., that alignment should not introduce new structural knowledge.

8.3.26 LogMap (*University of Oxford*)

LogMap is a scalable logic-based ontology matching system (Jiménez-Ruiz and Cuenca Grau 2011). It can deal with ontologies of hundreds of thousands of entities and billions of mappings.

The LogMap matching process is organised as in Fig. 8.9:

- *Indexing*: LogMap builds inverted indices for the labels as well as for their variations (such as synonyms) found in WordNet or UMLS. Extended class hierarchies, produced with the help of reasoners, such as HermiT and Condor, are also

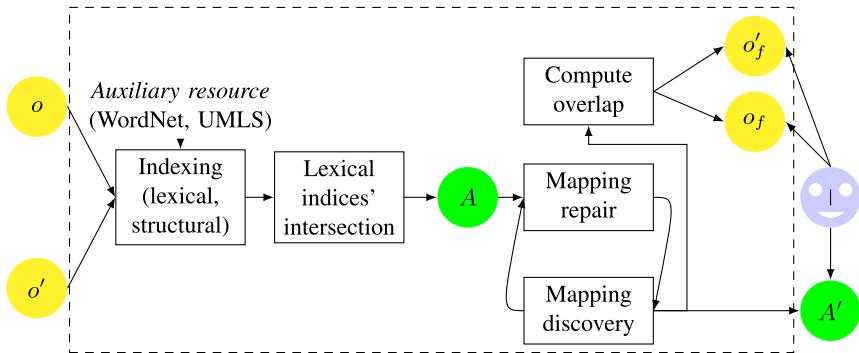


Fig. 8.9 The LogMap matching process.

indexed. Specifically, an interval labelling schema is used, which is an optimised data structure for storing directed acyclic graphs and trees.

- *Computation of initial anchor mappings*: finding an initial set of equivalences by intersecting, through exact string matching, the lexical indices of the ontologies to be matched.
- *Mapping repair*: running a sound and possibly incomplete reasoning to identify unsatisfiable classes with respect to the merge of the input ontologies and the mappings computed so far (see Sect. 7.8.2). Specifically, extended hierarchies and existing mappings are encoded into Horn propositional logic and the satisfiability of the encoding is tested by using the Dowling-Gallier algorithm (Dowling and Gallier 1984). A repair is computed for each unsatisfiable class through a greedy diagnosis algorithm.
- *Mapping discovery*: using the SMOA measure (Sect. 5.2.1) to match classes in the extended class hierarchies starting from the initial anchor mappings. LogMap iteratively alternates the mapping repair and the mapping discovery steps.
- *Overlap estimation*: computing fragments of each ontology which are overlapping, such that users can inspect presumably only these fragments when looking for the missing mappings.

LogMap2 (Jiménez-Ruiz et al. 2012) provides support for user interaction during the matching process. In an assisted mode, it operates in two phases:

- *computation of candidate mappings* to maximise recall. It starts with lexical matching supported by the indexing scheme of its predecessor. Then, it computes modules of the ontologies (Sect. 7.1.1) that encode the meaning of the entities involved in the candidate mappings, which are further used (instead of entire ontologies) for reasoning-based mapping error diagnosis.
- *assessment of candidates* to maximise precision without losing recall. This is done by discarding candidates through high thresholds as well as by path matching. The mapping repair functionality over modules (to improve performance) is also used to ultimately produce what is called reliable mappings.

User interaction is used for accepting or rejecting the uncertain mappings by presenting a reasonable number of such questions to users with reasonable delay between these.

8.3.27 PARIS (*Inria, Télécom ParisTech*)

PARIS is an automatic system for Probabilistic Alignment of Relations, Instances and Schema of (large) ontologies in RDFS (Suchanek et al. 2012). Instance- and schema-level links are put in interplay to enhance the overall entity (comprising classes, relations and instances) alignment quality. As the system name indicates, the underlying approach is probabilistic. Hence, confidence in correspondences is based on probability estimates, where logical rules used to model equality are transformed into probability assessments. It is assumed that the ontologies are available in their deductive closure, namely that all implications were explicitly added. It is also assumed that single ontologies do not contain equivalent entities in themselves. The system does not require any parameter tuning and computes equivalence and subclass relations. For example, the probability of a subclass (\leq) relation between two classes c and c' is proportional to the number of instances of c that are also instances of c' . In turn, the probability of a subrelation (\leq) between two relations r and r' is proportional to the number of pairs $\langle x, y \rangle$ in r (namely that r holds between the entities x and y) that are also pairs in r' . Initially the probability of equivalences between instances and then for subrelations is computed through a fixed point computation. Then, from the final assignment the equivalence between classes is produced. The probability of equivalence of two literals is set directly, e.g., for two strings being inversely proportional to the respective edit distance (Sect. 5.2.1).

8.4 Metamatching Systems

Metamatching systems are systems whose originality is in the way they use and combine other matching systems rather than in the matchers themselves.

8.4.1 APFEL (*University of Karlsruhe and University of Koblenz-Landau*)

APFEL (Alignment Process Feature Estimation and Learning) is a machine learning approach that explores user validation of initial alignments for optimising automatically the configuration parameters of some of the matching strategies of the system, e.g., weights, thresholds, for the given matching task (Ehrig et al. 2005). It is a component of the FOAM framework (Sect. 10.3.7). The overall architecture of APFEL is given in Fig. 8.10.

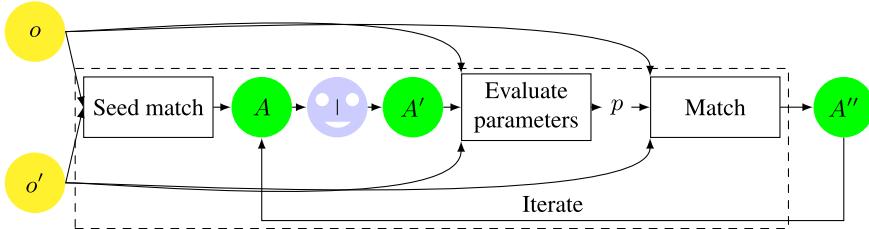


Fig. 8.10 APFEL architecture (adapted from (Ehrig 2007)): it generates alignments and asks users for feedback. Then it adjusts methods and aggregation parameters in order to minimise the error and iterate, if necessary.

APFEL parameterises the FOAM steps by using declarative representations of the (i) features engineered (Q_F); (ii) similarities assessed (Q_S); (iii) weight schemas, e.g., for similarity aggregation (Q_W); and (iv) thresholds (Q_T). For that purpose, the interfaces of matching systems are unified as Parameterisable Alignment Methods (PAM), which accept these parameters. First, given a matching system, for example QOM (Sect. 8.3.3), a PAM is initialised with it, e.g., PAM(QOM). Then, once an initial alignment is obtained, this alignment is validated by users. Finally, by analysing the validated alignment and the above parameters, with the help of machine learning techniques (Sect. 7.5), e.g., decision tree learner, neural networks, support vector machines of the Weka machine learning environment (Sect. 7.5), a tuned weighting scheme and thresholds are produced for the given matching task. This process can be iterated.

8.4.2 LCS (Queen's University Belfast)

LCS is a linguistic combination system for ontology matching (Ji et al. 2006). It handles ontologies in RDF and OWL and computes one-to-one alignments. The approach is based on ordered weighted average (Sect. 7.4.1) and linguistic quantifiers used to determine weights (Sect. 7.4.1). The weights are assigned not to matchers, but to a particular ordered position. LCS operates in four steps: (i) reorder the similarities provided by basic matchers in descending order; (ii) define a linguistic quantifier; (iii) compute the weights through the linguistic quantifier; (iv) apply the weights to aggregate the basic matcher results. The similarity computed by a matcher is viewed as a degree to which this matcher is satisfied for a pair of ontology entities. Various weight computation formulas have been defined as nondecreasing proportional fuzzy quantifiers inspired by linguistic quantifiers (see also Sect. 7.4.1). Examples of such linguistic quantifiers include *at least half*, namely that it satisfies at least half the matchers, *most*, namely that most matchers are satisfied, or *as many as possible*, namely that it satisfies as many matchers as possible. The system was evaluated with various basic matchers, looking for name, taxonomy, domain and range similarities and with different linguistic quantifiers.

8.4.3 Besana and Robertson (*University of Edinburgh*)

Besana and Robertson proposed a framework that aims at reducing the scope of matching tasks in open and distributed environments, such as peer-to-peer (Sect. 1.4) or multiagent (Sect. 1.6.1) systems, through service choreography statistics (Besana and Robertson 2007). In particular, the lightweight coordination calculus (LCC) was used as a communication language to implement the interactions among peers (Robertson 2004). The key idea is that the usage of certain terms in exchanged messages can be predicted by analysing the history of interactions between peers. The background intuition is that interactions that aim at performing the same tasks, such as booking a hotel, tend to be repetitively similar at each occurrence. In fact, different travel agencies would perform interactions with a customer on booking a hotel in a similar fashion. Thus, such repetitive interactions lead to patterns that are likely to be repeated in future when similar situations arise. Specifically, a probability distribution for a class of interactions is computed, which is then used as a basis for predicting the content of messages in future interactions. This aims at focussing the matching process only on the fragments of the underlying peer ontologies relevant to the interaction. These suggested terms, which are presumed to be used in the exchange messages in new interactions, are to be further matched by other ontology matchers. If the actual correspondences turn out to be correct, this avoids matching terms and more generally fragments of the peer ontologies, that are unrelated to the interaction, thereby reducing the scope and of the matching task, which is particularly crucial for dynamic applications. More specifically, this is meant to reduce the possible ambiguity of the terms to be matched, as well as to increase the efficiency of the matching process. Experiments showed that after less than a hundred of interactions, the proposed model consistently provides reliable suggestions.

8.4.4 eTuner (*University of Illinois and The MITRE Corporation*)

eTuner (Sayyadian et al. 2005; Lee et al. 2007) is a system which, given a particular matching task, automatically tunes a schema matching system (computing one-to-one alignments). For that purpose, it chooses the most effective basic matchers, and the best parameters to be used, e.g., thresholds. eTuner models a matching system as a triple: $\langle L, G, K \rangle$, such that

- L is a library of matching components, including basic matchers, e.g., edit distance, n -gram (Sect. 5.2.1), combiners, e.g., modules taking average, minimum and maximum of the results produced by basic matchers, constraint enforcers, e.g., pre-defined domain constraints or heuristics which are computationally expensive to be used as basic matchers, and match selectors, e.g., modules applying thresholds for determining the final alignment (Sect. 7.7.1).
- G is a directed graph which encodes the execution flow among the components of the given matching system.

- K is a set of *knobs* to be set (and named *knob configuration*). Matching components are viewed as black boxes which expose a set of adjustable knobs, such as thresholds, weights, or coefficients.

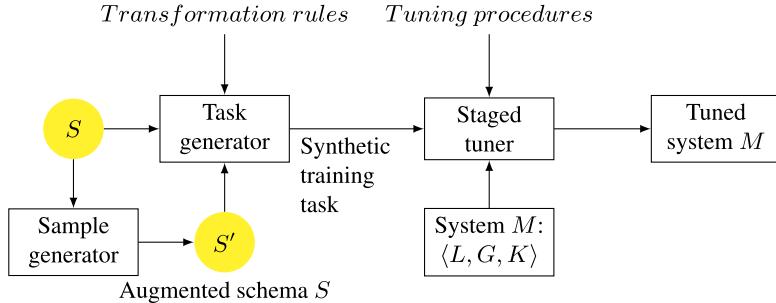


Fig. 8.11 eTuner architecture: eTuner generates a set of schemas to match with an initial schema. Then, it generates a plan for learning parameters. Finally, it tunes the method parameters and aggregation parameters.

The system works in two phases (see Fig. 8.11). During the first phase, a training task is synthesised with a known reference, given a single schema S . The system synthesises several schemas (S_1, S_2, \dots, S_n) out of S by altering it (for instance by modifying names of attributes, e.g., authors becomes aut). Thus, by taking a set of pairs $\{\langle S, S_1 \rangle, \langle S, S_2 \rangle, \dots, \langle S, S_n \rangle\}$ together with the reference correspondences available for free by construction of the synthetic schemas, the F-measure (Sect. 9.3) can be computed over any knob configuration. The second phase consists of searching the best parameters. Since the space of knob configurations can be large, the system uses a sequential, greedy approach, called *staged tuning*. In particular, by using the synthetic training task, it first tunes each of the basic matchers in isolation. Then, it tunes the combination of the basic matchers and the combiner, having the knobs of the basic matchers fixed (see also Sect. 7.6). Once the entire system is tuned, it can be applied to match schema S with any subsequent schemas.

8.4.5 mSeer (University of Wisconsin-Madison, The MITRE Corporation)

mSeer (match seer) is an approach for improving mediated schemas in data integration applications in order to facilitate finding correspondences with local or source schemas (Chai et al. 2008). It focusses on 1:1 alignments between relational sources. It introduced the notion of a *matchability score*, computed via a synthetic training task, which quantifies how well on average a given schema matches future schemas with a particular matching system. Thus, given a mediated schema (S), presumable future schemas (T) that will be matched against a

mediated schema, as well as a matching tool (M), mSeer computes a matchability score $m(S, T, M)$ based on F-measure (Sect. 9.3). Technically, it adapts eTuner (Sect. 8.4.4) for establishing T from S through a synthetic training task, e.g., by changing the employees table name to `emp`, and by replacing ,000 with K in the data values of the salary column. Using the matchability score, different types of matching mistakes, such as missing a correspondence or predicting a wrong correspondence, are analysed, e.g., by looking into the most likely general reasons leading to them. Once the matchability score has been computed for all the entities, these entities are ordered by increasing score. A *matchability report* is generated for each entity by grouping incorrect matches based on their underlying (general) reasons and by displaying (i) the reason, (ii) an example to illustrate the reason, and (iii) a suggestion of revisions to be made, thereby guiding users in revising the correspondences by addressing the reported mistakes. Finally, the system proposes changes to the mediated schema, such as renaming attributes or reformatting data values, still preserving its original semantics, while making it presumably easier to match with. Specifically, a revised mediated schema is automatically generated and users are then free to inspect (accept, reject, modify) the suggested revisions.

8.4.6 GOALS (*Polytechnic of Porto*)

GOALS is the Gecad Ontology ALignment System (Maio and Silva 2009). It implements a plug-in incremental and flexible approach to prototyping of complex matching systems without being committed to any predefined system architecture. The key building blocks of the system are (i) data structures, such as ontologies, which are manipulated by (ii) components (basic matchers) that are configurable, and used in (iii) a workflow specification to build complex matchers or metamatchers, that are finally run by (iv) an execution engine. Components, such as Falcon-AO (Sect. 8.3.9) and FOAM (Sect. 10.3.7), are independent from the system and are plugged-in through adapters. The metamatchers are configurable via an XML file. So, components can be added, changed or removed in order to improve over the previous results or one matching system can be used as a component in another more complex matching system. An example of a metamatcher is as follows. Load two ontologies in OWL. Match the input ontologies in parallel with the help of the following components: Jaro–Winkler (Sect. 5.2.1), WordNet (Sect. 5.2.2), Falcon (Sect. 8.3.9) and ClassStructure from the Alignment API (Sect. 10.1.5). The results of the first two components are combined through the maximum aggregator and the results of the second two components are combined through ordered weighted average (see Sect. 7.4.1). The results of these two aggregations are in turn combined through a weighted average (Sect. 7.4.1), while a threshold filter (Sect. 7.7.1) is used to produce the final alignment.

8.4.7 ContentMap (*Universitat Jaume I, University of Oxford*)

ContentMap is a logiC-based ONtology inTEgratioN Tool using MAPings (Jiménez Ruiz et al. 2009). It aims at helping users understand and evaluate the consequences of the integration of two ontologies as well as identify and handle possible errors. The proposed approach includes four main parts:

- *Computation and selection of alignments.* The system handles ontologies in OWL and allows for selecting matching tools, such as OLA (Sect. 8.3.8), AROMA (Sect. 8.3.14) or CIDER (Sect. 8.1.35), to generate alignments, which can be further filtered or edited manually.
- *Computation of new entailments* that hold in the merged ontology, identified through deductive differences between the two matched ontologies.
- *Detection of errors*, namely of unintended entailments, which should not be entailed in the merged ontology.
- *Repair of errors* through repair plans such that the unintended entailments are removed, while the desired ones still hold. Repair plans may be automatically selected based on confidence measures (see also Sect. 7.8.2). In turn, this helps in improving alignment quality.

The system was implemented as a Protégé plug-in and is equipped with a graphical user interface that helps users understand correspondences, accept or reject them, or filter them based on confidence thresholds. It also shows new entailments, the respective justifications (Sect. 10.2.2) and the available repair plans for the unintended entailments. Users can customise the kinds of entailments that should be invalidated or that should necessarily hold.

8.4.8 SMB (*Technion Israel Institute of Technology*)

SMB (Schema Matcher Boosting) is an approach to combining matchers into ensembles (Marie and Gal 2008; Gal 2011). It is based on a machine learning technique called boosting, that is able to select (presumably the most appropriate) matchers that participate in an ensemble. The approach re-uses about 30 matchers from OntoBuilder (Sect. 8.1.10). It also uses the *statistical monotonicity* principle when dealing with similarities. Specifically, a matcher is said to be statistically monotonic with respect to a matching task if its resulting similarity increases with precision. In turn, statistically monotonic matchers, implemented as classifiers, only slightly correlate with the true classification and their results are at least slightly better than a random choice.

Boosting strengthen weak classifiers by re-weighting the importance of elements in the training data set (Sect. 7.6.1). Specifically, the AdaBoost algorithm is used instead of least-square linear regression as in LSD (Sect. 8.2.4). SMB initiates greedily by selecting matchers (from a pool) that provide correct correspondences to a large part of the problem. Then, the ensemble is extended by adding matchers that solve

more specific cases. In the conducted experiments SMB ultimately used 24 out of 31 matchers.

8.4.9 AMC (SAP Research, University of Leipzig)

AMC (Auto Mapping Core) is a metamatching framework for rapid building and tuning of matching approaches (Peukert et al. 2011). It takes as input a pair of schemas and produces a mapping as output. It has an extensible library of diverse operators. These are:

- *Matcher* operators, which constitute the key part of the library and compute mappings among two schemas. Examples include the LMO matcher of Falcon (Sect. 8.3.9) or the NamePath matcher of COMA (Sect. 8.1.12).
- *Combination* operators, which combine multiple similarities into a single one, e.g., through a weighted average (Sect. 7.4.1).
- *Selection* operators, which filter mappings, e.g., by using thresholds (Sect. 7.7).
- *Analyser* operators, which define conditions for executing or not the subprocesses in order to adapt to a particular matching problem. For example, if a linguistic matcher provides sufficient quality, then no structure matching is performed as implemented in Falcon (Sect. 8.3.9).
- *Blocking* operators, which are used to reduce the search space and further compute mappings between possibly intersecting and smaller blocks. Examples include the partitioning method of Falcon (Sect. 8.3.9) or the fragmentation method of COMA (Sect. 8.1.12).

AMC provides a Matching Process Designer that visually supports building matching processes (Sect. 11.1), which can be executed and debugged with the help of breakpoints. A matching process is viewed as a directed acyclic graph that represents the execution order of the above mentioned operators. The operators can be dragged and connected visually. The currently executed operator is highlighted and intermediate results are visualised with the help of mapping cubes, where the axes represent source elements, target elements, and the respective similarity values. Such a representation helps individuating possible thresholds for the selection operators. Finally, AMC integrates as plug-ins the COMA++ (Sect. 10.3.2), Falcon (Sect. 8.3.9) and Rondo (Sect. 10.3.1) systems which are reused as specific operators.

8.4.10 AMS (SAP Research, Dresden University of Technology, University of Leipzig)

AMS is an Adaptive Matching System (Peukert et al. 2012). Its approach is based on the analysis of *features* computed from the input schemas and from the intermediate matching results. These features provide suggestions for the adaptation process. For example, the lengths of paths in a schema tree can suggest when to use a

path matcher (Sect. 5.2.1). In turn, intermediate matching results from a similarity matrix, such as low similarities, also viewed as ‘noise’, can negatively affect their future aggregations, and thus, it may be advisable to turn them to zero. Based on these features, several matching *rules* are applied in order to automatically build and adapt a matching process. The matching process is represented as a directed acyclic graph, whose nodes stand for operations to be executed, such as matching or aggregation. Rules are made of (i) a pattern that indicates a part of the graph where the rule should be applied, (ii) an action that applies changes to the patterns, (iii) a relevance function that indicates the relevance of the rule for the current process, (iv) a check function that rates the quality of changes introduced by an action. The system selects and runs rules based on their relevance to the current matching problem. Different types of rules are introduced. Some examples include *starting rules*, which add basic matchers, such as the *DataType* matcher, *aggregation rules*, which add various aggregations of matching results, such as average or ordered weighted average (Sect. 7.4.1), *rewrite rules*, which rewrite a non-empty process into a new one by changing its structure, e.g., noise reduction rule. The system provides libraries of several dozens of feature analysers and of rules. The whole matching process is iteratively extended and can also be edited by users.

8.5 Summary

The survey of systems considered in this chapter has multiplied the diversity of basic and advanced techniques of Chap. 5 and Chap. 6 by the variety of strategies for combining them introduced in Chap. 7. Moreover, usually each individual system innovates on a particular aspect. However, there are several constant features that are shared by the majority of systems.

In summary, the following can be observed concerning the presented systems:

- We have reviewed about 100 systems in this chapter, most of which, namely about 50 % are schema-based; about 25 % are mixed, i.e., rely both on schema- and instance-level information. Compared to the first edition of the book, the number of schema-based systems has doubled, of mixed systems has grown 1.5 times, of metamatching systems has quadrupled, while only a few new instance-based systems have appeared. We conclude that schema-based matching solutions have been so far investigated more intensively than instance-based or mixed solutions. In turn, investigation of metamatching systems have gained particular interest recently. We believe that this is an objective trend, since we have striven to cover state-of-the-art systems without bias towards any particular kind of solution. This is facilitated by the availability of other systems which may not have been the case in 2007.
- Most of the systems under consideration focus on specific application domains, such as books, music and biomedicine, as well as on dealing with particular ontology types, such as DTDs, relational schemas and OWL ontologies. Only few systems aim at being general, i.e., suit various application domains, and

generic, i.e., handle multiple types of ontologies. Some examples of the latter include Cupid (Sect. 8.1.11), COMA and COMA++ (Sect. 8.1.12), Similarity flooding (Sect. 8.1.14), S-Match (Sect. 8.1.18), GeRoMeSuite (Sect. 8.1.30) and Anchor-Flood (Sect. 8.1.40).

- Most of the approaches take as input a pair of ontologies, while only few systems take as input multiple ontologies. Some examples of the latter include DCM (Sect. 8.1.27), HSM (Sect. 8.1.28), Wise-Integrator (Sect. 8.3.6), and IceQ (Sect. 8.3.7).
- Most of the approaches handle only tree-like structures, while only few systems handle graphs. Some examples of the latter include Cupid (Sect. 8.1.11), COMA and COMA++ (Sect. 8.1.12), OLA (Sect. 8.3.8) and GeRoMeSuite (Sect. 8.1.30).
- Most of the systems focus on the discovery of simple one-to-one correspondences, though only few systems have tried to address the problem of discovering more complex correspondences, e.g., iMAP (Sect. 8.2.6), DCM (Sect. 8.1.27), HSM (Sect. 8.1.28), AOAS (Sect. 8.1.31), PORSCHE (Sect. 8.1.38), Agreement-Maker (Sect. 8.1.42) and Optima & Optima+ (Sect. 8.3.20).
- Most of the systems focus on computing confidence measures in the [0 1] range. They also most of the time return correspondences asserting equivalence relations. Only few systems compute different relations between ontology entities, such as disjointness or subsumption. Some examples of the latter include Ctx-Match (Sect. 8.1.17), S-Match (Sect. 8.1.18), DSSim (Sect. 8.1.44), TaxoMap (Sect. 8.1.46) and CSR (Sect. 8.3.21).
- Only several systems have introduced new ways of encoding the matching process, e.g., through Markov networks, such as iMatch (Sect. 8.1.47) and CODI (Sect. 8.3.25), or by proposing interesting interplays between data interlinking and schema matching, such as ILIADS (Sect. 8.3.15) and PARIS (Sect. 8.3.27), or by analysing in the matching process also image data, such as VSBM and GBM (Sect. 8.2.14).
- Many recent systems, such as Scarlet (Sect. 8.1.32), OMviaUO (Sect. 8.1.33) and BLOOMS & BLOOMS++ (Sect. 8.1.34), went beyond using WordNet as a source of background knowledge, for instance, by using Wikipedia and upper-level ontologies.
- Several recent systems have introduced alignment verification or debugging into the matching process, such as ASMOV (Sect. 8.3.17), Lily (Sect. 8.1.41), YAM++ (Sect. 8.3.23) and LogMap (Sect. 8.3.26).
- There is a growing number of systems that declare explicitly being able to handle efficiently large-scale ontologies, that is of tens of thousands of entities, e.g., Falcon (Sect. 8.3.9), Anchor-Flood (Sect. 8.1.40), Lily (Sect. 8.1.41), AgreementMaker (Sect. 8.1.42), LogMap (Sect. 8.3.26) and FSM (Sect. 8.2.13).
- Most of the systems are not equipped with a graphical user interface. Some examples of those with a graphical user interface include COMA++ (Sect. 8.1.12), S-Match (Sect. 8.1.18), AgreementMaker (Sect. 8.1.42), DSSim (Sect. 8.1.44), SAMBO (Sect. 8.3.13) and YAM++ (Sect. 8.3.23).

Table 8.1 summarises how matching systems cover the solution space in terms of the classifications of Chap. 4. For example, S-Match (Sect. 8.1.18) exploits string-based element-level matchers, external matchers based on WordNet, propositional satisfiability techniques, etc. OLA (Sect. 8.3.8), besides string-based element-level matchers, also exploits a matcher based on WordNet, iterative fixed point computation, etc. Table 8.1 also testifies that ontology matching research was mainly focussed on syntactic techniques so far. In fact, many systems rely on the same string-based techniques. Similar observation can also be made concerning the use of WordNet as an external resource of common knowledge. In turn, semantic techniques are only exploited in a few systems, e.g., CtxMatch (Sect. 8.1.17), S-Match (Sect. 8.1.18), or LogMap (Sect. 8.3.26). Concerning instance-based systems, techniques based on naive Bayes classifiers and common value patterns are the most prominent.

Table 8.1 Techniques used by the different systems.

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
DELTA Sect. 8.1.1	String-based	–	–	–
Hovy Sect. 8.1.2	String-based, Language-based	–	Taxonomic structure	–
TransScm Sect. 8.1.3	String-based, Built-in thesaurus	–	Taxonomic structure, Matching of neighbourhood	–
DIKE Sect. 8.1.4	String-based, Domain compatibility, WordNet	–	Matching of neighbourhood	–
SKAT Sect. 8.1.5	String-based, Auxiliary thesaurus, Corpus-based	–	Taxonomic structure, Matching of neighbourhood	–
Artemis Sect. 8.1.6	Domain compatibility, Language-based, Common thesaurus	–	Matching of neighbours via thesaurus, Clustering	–
H-Match Sect. 8.1.7	Domain compatibility, Language-based, Domains and ranges, Common thesaurus	–	Matching of neighbours via thesaurus, Relations	–
Tess Sect. 8.1.8	String-based, domain compatibility	–	Matching of neighbours	–

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
Anchor-Prompt Sect. 8.1.9	String-based, Domains and ranges	–	Bounded paths matching: (arbitrary links), Taxonomic structure	–
OntoBuilder Sect. 8.1.10	String-based, Language-based, Thesaurus look-up	–	–	–
Cupid Sect. 8.1.11	String-based, Language-based, Data types, Key properties, Auxiliary thesauri	–	Tree matching weighted by leaves	–
COMA/COMA++ Sect. 8.1.12	String-based, Language-based, Data types, Auxiliary thesauri	Alignment reuse	DAG (tree) matching with a bias towards various structures, e.g., leaves, Repository of structures	–
QuickMig Sect. 8.1.13	String-based	Domain ontology	–	–
Similarity flooding Sect. 8.1.14	String-based, Data types, Key properties	–	Iterative fixed point computation	–
XClust Sect. 8.1.15	Cardinality constraints, WordNet	–	Paths, Children, Leaves, Clustering	–
MapOnto Sect. 8.1.16	–	External alignments	Structure comparison	–
CtxMatch Sect. 8.1.17	String-based, Language-based, WordNet	–	–	Based on description logics
S-Match Sect. 8.1.18	String-based, Language-based, WordNet	–	–	Propositional SAT
HCONE Sect. 8.1.19	Language-based, (LSI), WordNet	–	–	–
MoA Sect. 8.1.20	Language-based, WordNet	–	–	–
ASCO Sect. 8.1.21	String-based, Language-based, WordNet	–	Iterative similarity propagation	–
Stroulia & Wang Sect. 8.1.22	String-based, Language-based, WordNet	–	Structure comparison	–

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
MWSDI Sect. 8.1.23	String-based, Language-based, WordNet	–	Structure comparison	–
SeqDisc Sect. 8.1.24	String-based, Language-based	–	Leafs, Children, Ancestor comparison	–
BayesOWL Sect. 8.1.25	Text classifier, Google	–	Bayesian inference	–
OMEN Sect. 8.1.26	–	External alignment	Bayesian inference, Metarules	–
DCM Sect. 8.1.27	–	–	Correlation mining, Statistics	–
HSM Sect. 8.1.28	–	–	Co-occurrence patterns, Statistics	–
CBW Sect. 8.1.29	String-based	–	Coincidence-based weighting	–
GeRoMeSuite Sect. 8.1.30	String-based	–	Similarity flooding, Children	–
AOAS Sect. 8.1.31	String-based, Language-based	UMLS	Compatible, <i>is-a, part-of</i> paths	Rule-based inference
Scarlet Sect. 8.1.32	String-based	Multiple on-line ontologies	–	Ad hoc rule-based inference
OMviaUO Sect. 8.1.33	String-based, Language-based	SUMO- OWL, OpenCyc, DOLCE	Taxonomic structure	Rule-based inference
BLOOMS/ BLOOMS+ Sect. 8.1.34	Language-based, Alignment API	–	Taxonomic structure	Rule-based inference
CIDER Sect. 8.1.35	String-based, Language-based	Swoogle ontologies	Structure comparison	Transitive inference
Elmeleegy & al. Sect. 8.1.36	Data type compatibility, Query logs	–	Similarity flooding	–
BeMatch Sect. 8.1.37	String-based, Language-based, WordNet	–	Graph isomorphism	–
PORSCHE Sect. 8.1.38	String-based, Language-based, Domain thesauri	–	Clustering, Tree mining	–
MatchPlanner Sect. 8.1.39	SecondString, Language-based, WordNet	–	Neighbor profiles	–

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
Anchor-Flood Sect. 8.1.40	String-based, Language-based, WordNet	–	Internal, external similarities, Iterative anchor-based similarity propagation	–
Lily Sect. 8.1.41	String-based	–	Variations of similarity flooding	–
Agreement-Maker Sect. 8.1.42	String-based, Language-based, WordNet	–	Descendant, sibling similarities	–
Homolonto Sect. 8.1.43	Language-based, Word occurrence	–	Children similarity	–
DSSim Sect. 8.1.44	String-based, Language-based, WordNet	–	Graph similarity based on leaves	Rule-based fuzzy inference
MapPSO Sect. 8.1.45	String-based, Language-based, WordNet	–	Population-based optimisation	–
TaxoMap Sect. 8.1.46	String-based, Language-based	–	Structure comparison via <i>is-a</i> hierarchies	–
iMatch Sect. 8.1.47	String-based	External alignment	Probabilistic (Markov) reasoning, Metarules	–
T-tree Sect. 8.2.1	–	–	Correlation mining	–
CAIMAN Sect. 8.2.2	String-based (Rocchio classifier)	–	–	–
FCA-merge Sect. 8.2.3	–	–	Formal concept analysis	–
LSD/GLUE/ iMAP Sect. 8.2.4– 8.2.6	WHIRL, Naive Bayes, Domain constraints	–	Hierarchical structure	–
Automatch Sect. 8.2.7	Naive Bayes	–	Internal structure, Statistics	–
SBI&NB Sect. 8.2.8	Statistics, Naive Bayes	–	Pachinko Machine, Naive Bayes	–
Kang & Naughton Sect. 8.2.9	Information entropy	–	Mutual information, Dependency graph matching	–
Dumas Sect. 8.2.10	String-based, WHIRL	–	Instance identification	–
Wang & al. Sect. 8.2.11	Language-based	–	Mutual information	–

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
sPLMap Sect. 8.2.12	Naive Bayes, kNN classifier, String-based	–	–	–
FSM Sect. 8.2.13	String-based	–	–	–
VSBM& GBM Sect. 8.2.14	Statistics, SVM	–	Correlations in a multimedia graph	–
ProbaMap Sect. 8.2.15	Statistics, Naive Bayes, C4.5, SVM	–	–	–
SEMINT Sect. 8.3.1	Neural network, Data types, Value patterns	–	–	–
IF-Map Sect. 8.3.2	String-based	–	Formal concept analysis	–
NOM/QOM Sect. 8.3.3	String-based, Domains and ranges, Application vocabulary	–	Matching of neighbours, Taxonomic structure	–
oMap Sect. 8.3.4	Naive Bayes, String-based	–	Similarity propagation	–
Xu & al. Sect. 8.3.5	String-based, Language-based, WordNet	Domain ontology	Decision trees	–
Wise- Integrator Sect. 8.3.6	Language-based, String-based, Data types, Value patterns, WordNet	–	Clustering	–
IceQ Sect. 8.3.7	String-based, Value patterns, Domain dictionary	–	Clustering	–
OLA Sect. 8.3.8	String-based, Language-based, Data types, WordNet	–	Iterative fixed point computation, Matching of neighbours, Taxonomic structure	–
Falcon-AO Sect. 8.3.9	String-based, WordNet	–	Structural affinity	–
RiMOM Sect. 8.3.10	String-based, Naive Bayes, WordNet	–	Taxonomic structure, Similarity propagation	–

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
Corpus-based matching Sect. 8.3.11	String-based, Language-based, Naive Bayes, Value patterns, Domain constraints	—	Corpus schemas	—
iMapper Sect. 8.3.12	Linguistic classifier, WordNet	—	Linguistic classifier, Structure comparison	—
SAMBO Sect. 8.3.13	String-based, Naive Bayes, WordNet	UMLS	Iterative structural similarity base on <i>is-a</i> , <i>part-of</i> hierarchies	—
AROMA Sect. 8.3.14	String-based	—	Association rules	—
ILIADS Sect. 8.3.15	String-based, Language-based, WordNet	—	Matching neighbours, Clustering	Rule-based inference
SeMap Sect. 8.3.16	String-based, Language-based, Instance distribution, WordNet	Domain ontology	Similarity flooding	—
ASMOV Sect. 8.3.17	String-based, Language-based, Object similarity, WordNet	UMLS	Iterative fixed point computation, Hierarchical, restriction similarities	Rule-based inference
HAMSTER Sect. 8.3.18	String-based, Language-based, Naive Bayes, Click logs	—	Structure comparison	—
Smart Matcher Sect. 8.3.19	COMA++, FOAM	—	Structure comparison	—
GEM/Optima+/Optima+ Sect. 8.3.20	String-based, Language-based, WordNet	—	Expectation-Maximisation, Matching of neighbours	—
CSR Sect. 8.3.21	Feature-based similarity, Machine learning	—	Feature-based similarity, Machine learning	—
Prior+ Sect. 8.3.22	String-based, Language-based	—	Feature-based similarity, Neural network	—
YAM/YAM++ Sect. 8.3.23	Machine learning, SecondString, SimMetrics, WordNet	—	Structure profiles, Similarity flooding	—

Table 8.1 (Continued.)

	Element-level		Structure-level	
	Syntactic	Semantic	Syntactic	Semantic
MoTo Sect. 8.3.24	Naive Bayes, Neural network, k -Nearest neighbour	–	Structural validation: taxonomy, Other relations	–
CODI Sect. 8.3.25	SimMetrics	–	Structure comparison	Markov net inference
LogMap/ LogMap2 Sect. 8.3.26	String-based, Language-based, WordNet	UMLS	Structure comparison	Propositional Horn satisfiability
PARIS Sect. 8.3.27	String-based	–	Probabilistic estimates via iterative fixed point computation	–

Table 8.2 summarises the position of these systems with regard to some of the requirements of Table 3.1 (p. 59), namely, those requirements that can be given in the specification of the system rather than being measured. In Table 8.2, the *Input* column stands for the input taken by the systems. In particular, it mentions the languages that the systems are able to handle (if this information was not available from the articles describing the corresponding systems, we used general terms, such as database schema and ontology instead). This is, of course, very important for someone who has a certain type of ontology to match and is looking for a system. The *Needs* column stands for the resources that must be available for the system to work. This covers the automatic aspect of Sect. 3.2, which is here denoted by *user* when user feedback is required, *semi* when the system can take advantage of user feedback but can operate without it and *auto* when the system works without user intervention (of course, users can influence the system by providing the initial input or evaluating the results afterwards, but this is not taken into account here). Similarly, the *instances* value specifies that the system requires data instances to work. In addition, some systems may require *training* before the actual matching as well as *alignment* to be improved. The *Output* column denotes the form of the results given by the system: *Alignment* means that the system returns a set of correspondences, *merge* that it merges the input ontologies or schemas, *axioms* or *rules* that it provides rules for querying or completing the ontologies, etc.

The *Output* delivered by a system is very important because it shows the capacity of the system to be used for some applications, e.g., a system delivering views and data translators cannot be used for merging ontologies as is. It is remarkable that many systems deliver alignments. As such, they are not fully committed to any kind of operation to be performed and can be used in a variety of applications. This could be viewed as a sign of possible interoperability between systems. However, due to lack of a common alignment format (see Chap. 10), each system uses its own way to deliver alignments (as lists of URIs, tables, etc.). Finally, the *Operation* column describes the ways in which a system can process alignments.

Table 8.2 Position of the presented systems with regard to the requirements of Chap. 3.

System	Input	Needs	Output	Operation
DELTA Sect. 8.1.1	Relational schema, EER	User	Alignment	–
Hovy Sect. 8.1.2	Ontology	Semi	Alignment	–
TransScm Sect. 8.1.3	SGML, OO	Semi	Translator	Data translation
DIKE Sect. 8.1.4	ER	Semi	Merge	Query mediation
SKAT Sect. 8.1.5	RDF	Semi	Bridge rules	Data translation
Artemis Sect. 8.1.6	Relational schema, OO, ER	Auto	Views	Query mediation
H-Match Sect. 8.1.7	OWL	Auto	Alignment	P2P query mediation
Tess Sect. 8.1.8	Database schema	Auto	Rules	Version matching
Anchor- Prompt Sect. 8.1.9	OWL, RDF	User	Axioms (OWL/RDF)	Ontology merging
OntoBuilder Sect. 8.1.10	Web form, XML schema	User	Mediator	Query mediation
Cupid Sect. 8.1.11	XML schema, Relational schema	Auto	Alignment	–
COMA/ COMA++ Sect. 8.1.12	Relational schema, XML schema, OWL	User	Alignment	Data translation
QuickMig Sect. 8.1.13	COMA++	User, Sample data injection	Alignment, Mapping category	Data translation
Similarity flooding Sect. 8.1.14	XML schema, Relational schema	User	Alignment	–
XClust Sect. 8.1.15	DTD	Auto	Alignment	–
MapOnto Sect. 8.1.16	Relational schema, XML schema, OWL	Alignment	Rules	Data translation
CtxMatch/ CtxMatch2 Sect. 8.1.17	Classification, OWL	User	Alignment	–
S-Match Sect. 8.1.18	Classification, XML schema, OWL	Auto	Alignment	–
HCONE Sect. 8.1.19	OWL	Auto, Semi, User	Ontology	Ontology merging

Table 8.2 (Continued.)

System	Input	Needs	Output	Operation
MoA Sect. 8.1.20	OWL	Auto	Axioms, OWL	–
ASCO Sect. 8.1.21	RDFS, OWL	Auto	Alignment	–
Stroulia & Wang Sect. 8.1.22	WSDL	Auto	Alignment	–
MWSDI Sect. 8.1.23	WSDL, OWL	Auto	Alignment	–
SeqDisc Sect. 8.1.24	WSDL	Auto	Alignment	–
BayesOWL Sect. 8.1.25	Classification, OWL	Auto	Alignment	–
OMEN Sect. 8.1.26	OWL	Auto, Alignment	Alignment	–
DCM Sect. 8.1.27	Web form	Auto	Alignment	Data integration
HSM Sect. 8.1.28	Web form	Auto	Alignment	–
CBW Sect. 8.1.29	OWL	Auto	Alignment	–
GeRoMeSuite Sect. 8.1.30	SQL DDL, XML, OWL	Auto, Semi	Alignment	Merge, Compose
AOAS Sect. 8.1.31	OWL	Auto	Alignment	–
Scarlet Sect. 8.1.32	OWL	Auto	Alignment	–
OMviaUO Sect. 8.1.33	RDFS, OWL	Auto	Alignment	–
BLOOMS/ BLOOMS+ Sect. 8.1.34	RDFS, OWL	Auto	Alignment	–
CIDER Sect. 8.1.35	OWL	Auto, Training	Alignment	–
Elmeleegy & al. Sect. 8.1.36	Relational schema	Auto, Query logs	Alignment	–
BeMatch Sect. 8.1.37	BPEL, WCSL	Auto, Semi	Alignment	Service transformation
PORSCHE Sect. 8.1.38	XSD	Auto	Alignment	Mediated schema
Match- Planner Sect. 8.1.39	XML	Auto, Decision tree	Alignment	–

Table 8.2 (Continued.)

System	Input	Needs	Output	Operation
Anchor-Flood Sect. 8.1.40	RDFS, OWL	Auto	Alignment	–
Lily Sect. 8.1.41	OWL	Auto, Semi	Alignment	–
Agreement-Maker Sect. 8.1.42	XML, RDFS, OWL, N3	Auto, Semi	Alignment	–
Homolonto Sect. 8.1.43	OBO	Auto, Semi	Alignment,	Homologous groups
DSSim Sect. 8.1.44	OWL, SKOS	Auto	Alignment	Question answering
MapPSO Sect. 8.1.45	OWL	Auto	Alignment	–
TaxoMap Sect. 8.1.46	OWL	Auto, Semi	Alignment	–
iMatch Sect. 8.1.47	OWL	Auto, Semi	Alignment	–
T-tree Sect. 8.2.1	Ontology	Auto, Instances	Alignment	–
CAIMAN Sect. 8.2.2	Classification	Semi, Instances, Training	Alignment	–
FCA-merge Sect. 8.2.3	Ontology	User, Instances	Ontology	Ontology merging
LSD/GLUE Sect. 8.2.4, Sect. 8.2.5	Relational schema, XML schema, Taxonomy	Auto, Instances, Training	Alignment	–
iMAP Sect. 8.2.6	Relational schema	Auto, Instances, Training	Alignment	–
Automatch Sect. 8.2.7	Relational schema	Auto, Instances, Training	Alignment	–
SBI&NB Sect. 8.2.8	Classification	Auto, Instances, Training	Alignment	–
Kang & Naughton Sect. 8.2.9	Relational schema	Instances	Alignment	–
Dumas Sect. 8.2.10	Relational schema	Instances	Alignment	–
Wang & al. Sect. 8.2.11	Web form	Instances	Alignment	Data integration
sPLMap Sect. 8.2.12	Database schema	Auto, Instances, Training	Rules	Data translation
FSM Sect. 8.2.13	Database schema	Auto, Instances	Alignment	–
VSBM&GBM Sect. 8.2.14	Ontology	Auto, Instances, Training	Alignment	–

Table 8.2 (Continued.)

System	Input	Needs	Output	Operation
ProbaMap Sect. 8.2.15	Taxonomy	Auto, Instances, Training	Alignment	–
SEMINT Sect. 8.3.1	Relational schema	Auto, Instances (opt.), Training	Alignment	–
IF-Map Sect. 8.3.2	KIF, RDF	Auto, Instances, Common reference	Alignment	–
NOM/QOM Sect. 8.3.3	RDF, OWL	Auto, Instances (opt.)	Alignment	–
oMap Sect. 8.3.4	OWL	Auto, Instances (opt.), Training	Alignment	Query answering
Xu & al. Sect. 8.3.5	XML schema, Taxonomy	Auto, Instances (opt.), Training	Alignment	–
Wise- Integrator Sect. 8.3.6	Web form	Auto	Mediator	Data integration
IceQ Sect. 8.3.7	Web form	Auto, Semi	Alignment	–
OLA Sect. 8.3.8	RDF, OWL	Auto, Instances (opt.)	Alignment	–
Falcon-AO Sect. 8.3.9	RDF, OWL	Auto, Instances (opt.)	Alignment	–
RiMOM Sect. 8.3.10	OWL	Auto, Instances (opt.)	Alignment	–
Corpus-based matching Sect. 8.3.11	Relational schema, Web form	Text corpora, Instances, Training	Alignment	–
iMapper Sect. 8.3.12	RML	Auto, Semi, Documents	Alignment	–
SAMBO Sect. 8.3.13	OWL	Auto, Documents (opt.)	Alignment	Ontology merging
AROMA Sect. 8.3.14	Classification, OWL	Auto, Instances	Alignment	–
ILIADS Sect. 8.3.15	OWL	Auto, Instances (opt.)	Alignment	Ontology merging
SeMap Sect. 8.3.16	XML	Auto, Semi Instances (opt.)	Alignment	–
ASMOV Sect. 8.3.17	OWL	Auto, Instances (opt.)	Alignment	–
HAMSTER Sect. 8.3.18	XML	Auto, Instances (opt.), Clicklog	Alignment	–

Table 8.2 (Continued.)

System	Input	Needs	Output	Operation
Smart Matcher Sect. 8.3.19	UML	Auto, User instances (opt.)	Alignment	Instance transformation
GEM/Optima/ Optima+ Sect. 8.3.20	RDF, OWL, N3	Auto, Instances (opt.)	Alignment	–
CSR Sect. 8.3.21	OWL	Auto, Instances (opt.)	Alignment	–
Prior+ Sect. 8.3.22	OWL	Auto, Instances (opt.)	Alignment	–
YAM/YAM++ Sect. 8.3.23	XML, OWL	Auto, Semi	Alignment	–
MoTo Sect. 8.3.24	OWL	Auto, Instances (Opt.)	Alignment	–
CODI Sect. 8.3.25	OWL	Auto, Instances (Opt.)	Alignment	–
LogMap/ LogMap2 Sect. 8.3.26	OWL	Auto, Semi	Alignment, Ontologies overlap	–
PARIS Sect. 8.3.27	RDFS	Auto, Instances	Alignment	–
APFEL Sect. 8.4.1	RDF, OWL	User	Alignment	–
LCS Sect. 8.4.2	RDF, OWL	Auto	Alignment	–
Besana & Robertson Sect. 8.4.3	LCC	Auto, History of interactions	Suggested terms	–
eTuner Sect. 8.4.4	Relational schema, Taxonomy	Auto	Alignment	–
mSeer Sect. 8.4.5	Relational schema	Auto, Semi, Instances	Matchability score & report	Revised mediated schema
GOALS Sect. 8.4.6	OWL	Auto	Alignment	–
ContentMap Sect. 8.4.7	OWL	Auto, Semi	Alignment, Alignment repairs	Integrated ontology
SMB Sect. 8.4.8	Web form, XML schema, OWL	Auto	Alignment	–
AMC Sect. 8.4.9	Relational schema, XML, OWL	Auto, Semi, Instances (Opt.)	Alignment	–
AMS Sect. 8.4.10	Relational schema, XML, OWL	Auto, Semi, Instances (Opt.)	Alignment	–

Not all the requirements are addressed in Table 8.2. Indeed, completeness, correctness, run time should not be judged from the claims of system developers. No meaningful system can be proved to be complete, correct or as fast as possible in a task like ontology matching. Therefore, the degree of fulfilment of these requirements must be evaluated and compared across systems. Moreover, different applications have different priorities regarding these requirements. Hence, they may need different systems. Thus, this evaluation depends on an application in which the system is to be used.

It is difficult to evaluate and compare systems without commonly agreed test benches, principles and available implementations. The next chapter presents methods for empirical evaluation and comparison of matching systems.

Chapter 9

Evaluation of Matching Systems

The increasing number of methods available for ontology matching commands the evaluation of these methods.

Matching systems are difficult to compare, but the ontology matching field can only evolve if evaluation criteria are provided. These should help system designers assess the strengths and weaknesses of their systems as well as help application developers choose the most appropriate algorithm.

In this chapter, we first consider the main motivations for evaluating matching systems and the principles that could guide such an evaluation (Sect. 9.1). We also discuss existing evaluation resources, different available data sets and the structure of some of these data sets (Sect. 9.2). Then, we overview the measures used for the evaluation of matching systems (Sect. 9.3). Finally, we consider in more detail the settings of an evaluation protocol for a particular application, as opposed to evaluation for comparing matching systems in general (Sect. 9.4).

9.1 Evaluation Principles

All evaluation activities must be carried out with a clear procedure. So we first recall here the goal of evaluating ontology matching systems (Sect. 9.1.1), the principles on which evaluation should be based (Sect. 9.1.2) and some examples of evaluation initiatives (Sect. 9.1.3).

9.1.1 Goals

A major and long term purpose of the evaluation of ontology matching methods is to help designers and developers of such methods improve them and to help users evaluate the suitability of the proposed methods to their needs. The evaluation should thus be run over several years in order to allow for adequate measurement of the evolution of the field.

Evaluation should also help assess absolute results produced by the matching systems, i.e., what are the properties achieved by a system, and relative results, i.e., how these compare to the results of other systems.

One particular kind of evaluation is benchmarking. A benchmark is a well-defined set of tests on which the results of a system or subsystem can be measured (García Castro et al. 2004). It should enable the measure of the degree of achievement of proposed tasks on a well-defined scale (that can be achieved or not). It should be reproducible and stable, so that it can be used repeatedly for (i) testing the improvement or degradation of a system with certainty, (ii) situating a system among others.

A medium term goal for evaluation efforts is to set up a collection of reference sets of tests, or benchmark suites of the available tools and to compare tool evolution with regard to this reference. Building benchmark suites is highly valuable not just for those who participate in planned evaluations but for all the research community, since system designers can use these at any time and compare themselves to others.

9.1.2 Principles

We describe below several principles that must guide the evaluation process:

Systematic procedure. Evaluation results have to be nonambiguous and their procedure should be reproducible. Thus, the application of the procedure to different systems or to the same system at different moments of time should be comparable.

Continuity. Evaluation, and most particularly benchmarking, must not be a one-shot exercise but a continuous effort in order to identify the progress made by the field and eventually stop when no more progress is made anymore.

Quality and equity. The evaluation rules must be precise and defined beforehand. In order to be worthwhile, the evaluation material must be of the best possible quality. This also means that the evaluation material must not be biased towards a particular kind of algorithm but driven only by the tasks to be solved.

Dissemination. In order to have a high impact, the evaluation activity must be disseminated without excessive barriers. To that extent, the data sets and results produced must be published and made as freely available as possible. The evaluation campaigns must be open to participants worldwide.

Intelligibility. It is of high importance that the evaluation results could be analysed by the stakeholders and understood by everyone. For that purpose, it is useful not only that the final results are published but also the alignments themselves. Finally, of high importance is the archival explanation of the results to the stakeholders.

Evaluations are usually based on three successive steps (García Castro et al. 2004):

Planning involves defining the task to be performed as well as its constraints, e.g., resources allowed, computer environment, required output, finding data sets on which to perform the tasks and setting the measures to be used.

Processing consists of executing the plan.

Analysing evaluates the results achieved according to planned measurements.

These three steps can be complemented by a recalibration loop that helps in redefining the plan for the next evaluation from lesson learnt in the current one.

9.1.3 Examples of Evaluations

In order to illustrate what can be done as evaluation, we briefly discuss a model evaluation initiative, called TREC, and the Ontology Alignment Evaluation Initiative modelled upon it.

Text REtrieval Conference

TREC¹ is the Text REtrieval Conference organised by the National Institute of Standards and Technology (NIST) in the USA. It has been run yearly since 1992. It is a very good model for evaluation in a focussed computer science research field, especially because it has been very successful in helping the field to progress.

The goals of TREC are to:

- increase research in information retrieval-based on large-scale collections,
- provide a forum for stakeholders,
- facilitate technology transfer,
- improve the evaluation methodology,
- create a series of test collections on various aspects of information retrieval.

It is now organised in several tracks, each of which corresponding to one kind of evaluation, which, in turn, is organised over several years. Five years is now the accepted standard in order to be able to compare results. Tracks organised so far include

- static text retrieval,
- interactive retrieval,
- information retrieval in a narrow domain, e.g., genomics, using ad hoc resources,
- media retrieval (other than text), or
- answer finding.

Typically each track has between 8 and 20 participants. While each track is precisely defined, TREC has now a track record on investigating the evaluation of many different features of retrieval tasks.

¹<http://trec.nist.gov>.

Table 9.1 Overview of the OAEI campaigns (extended from (Euzenat et al. 2011)).

	year	location	#tests	#participants
I3CON	2004	Gaithersburg, US	10	5
OAC	2004	Hiroshima, JP	1	4
OAEI	2005	Banff, CA	3	7
OAEI	2006	Athens, US	6	10
OAEI	2007	Busan, KR	7	17
OAEI	2008	Karlsruhe, DE	8	13
OAEI	2009	Chantilly, US	9	16
OAEI	2010	Shanghai, CN	6	15
OAEI	2011	Bonn, DE	5	18
OAEI	2011.5	Heraklion, GR	5	19
OAEI	2012	Boston, US	7	23

Ontology Alignment Evaluation Initiative

Since 2004, a group of researchers on ontology matching have run several evaluation campaigns which are identified as the Ontology Alignment Evaluation Initiative (OAEI).² After a cold start in 2004, with two events, the Information Interpretation and Integration Conference (I3CON), held at the NIST Performance Metrics for Intelligent Systems (PerMIS) workshop, and the Ontology Alignment Contest at the third Evaluation of ONtology-based tools (EON) workshop, OAEI started under this name in 2005 and has been run yearly since then with an extra 2011.5 edition in 2012 (Euzenat et al. 2011). It has offered various tracks summarised in Table 9.1. Since 2009, it has also offered an instance matching track, which is not reported here.

Until 2011, in each of these campaigns, the participants were required to provide their resulting alignments in the Alignment format (Sect. 10.1.5). They are equipped with the Alignment API (Sect. 10.3.6) for helping them to produce and to assess the results before the meeting. Results to all tests are compulsory as well as a fixed-format paper describing experiences with tests processing. Participants were also expected to present their results at the meeting. The results of these tests are evaluated by clearly announced measures, typically precision and recall (Sect. 9.3). Finally, the results of evaluation campaigns as well as the full data sets are available for download from the OAEI web site.²

Since 2011, some of these evaluations have been automated through the SEALS platform (see Sect. 9.1.5). The OAEI campaigns tend to set a solid basis for evaluating the progress in matching algorithms by providing a stable benchmark suite, thus allowing progress to be monitored year after year and facilitating the calibration of the participating matching algorithms. A summary of these campaigns is available

²<http://oaei.ontologymatching.org>.

in (Euzenat et al. 2011; Aguirre et al. 2012) and full details are on the OAEI web site.

9.1.4 Types of Evaluations

In Chap. 2, we characterised an alignment as a set of pairs of entities e and e' , coming from each ontology o and o' , related by a particular relation r . In addition, many algorithms assign a confidence measure n to correspondences. From this characterisation it is possible to require any matching method to output an alignment (Noy and Musen 2002a; Euzenat 2003), given

- two ontologies to be matched,
- a partial input alignment, which may be possibly empty,
- a characterisation of the desired alignment, e.g., $1:+$, $?:\?$.

From this output, the quality of the matching process could be assessed with the help of measurements of the difference between the output and a reference alignment (see Fig. 9.1).

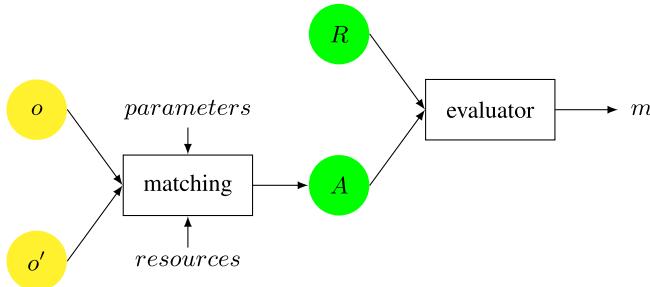


Fig. 9.1 Basic evaluation design: a matcher receives two ontologies o and o' as input and generates an alignment A using a fixed set of resources and parameters. An evaluation component receives this alignment and computes a (set of) quality measure(s) m —typically precision and recall—by comparing it to the reference alignment R .

From this basic setting, there are several ways of planning the evaluation depending, in part, of its purpose. There may be several classifications depending on the criteria used. Let us consider a classification of evaluations with regard to what they are supposed to evaluate:

Competence benchmarks allow the characterisation of the level of competence (and performance) of a particular system with regard to a set of well defined tasks. Usually, tasks are designed to isolate particular characteristics. This kind

of benchmarking is reminiscent to kernel benchmarks or unit tests, such as the Standard Performance Evaluation Corporation (SPEC) benchmarks.³

Competence benchmarks aim at characterising the kind of task each method is good for or the kind of input it can handle well. There are many different areas in which methods can be evaluated. One approach is to look at the kind of features they use for finding matching entities, for example, following one of the classifications of Chap. 4.

Benchmark suites must be stable so that they enable the monitoring of the evolution of the field over time. Moreover, they do not need to be run blindly since they are run several times. Thus, they can be freely distributed and designers of new systems can take advantage of them at any time.

Comparative evaluation allows for comparing the results of various systems or several versions of the same system on a common task. Since the goal is to compare systems, it is of the utmost importance that the rules and the evaluation criteria are clearly specified.

Because it is difficult to guarantee that the systems are not tuned for the evaluation, it is preferable to run blind tests or nearly blind tests. This means that the participants become aware of the data set very shortly before the evaluation and that the data set must be changed at each evaluation.

Finally, because it is run in a limited time span with relatively similar resources, such an evaluation requires a well defined processing mode. In counterpart, this allows acquiring more accurate nonfunctional measurements, such as run time and memory.

Application-specific evaluation allows for comparing the results of various systems on the output of a particular application instead of considering the alignments in isolation. Such evaluations are useful for a company that has a real application and wants to find the best system to use in this application. It can also be useful for a competitive evaluation.

The goals of these three kinds of evaluations are different. Competence benchmarks aim at helping system designers to evaluate their systems and to situate them with regard to a common stable framework. It is also helpful for improving individual systems. A comparative evaluation enables the comparison of systems on general-purpose tasks. Its goal is mainly to help the improvement of the field as a whole, rather than individual systems. It can also help users in selecting an appropriate system. Application-specific evaluations aim at identifying an adequate system for one particular application at one particular moment (see also Sect. 9.4).

9.1.5 Automation

The evaluation scheme presented in Sect. 9.1.4 (Fig. 9.1) suggests that by requiring matchers to implement a minimal interface, it would be possible to automate eval-

³<http://www.spec.org>.

ation. The SEALS (Semantic Evaluation at Large Scale) project⁴ experimented with this and automated evaluation to a large extent. One of its goals was to provide an infrastructure, the SEALS platform, for automatically executing evaluations. It has been experimented with in two ways in OAEI:

- In 2010, a web service-based infrastructure was set up and used for three data sets (Trojahn et al. 2010a): participants had to implement a web service interface at their premises and the platform submitted them tests and recorded the results. In this configuration, tool developers could trigger the evaluation and receive the results.
- In 2012 (OAEI 2011.5 and 2012), participants submitted their system packaged with a minimal Java interface to the platform in which the tools were run against the tests. The SEALS modality was used for all data sets on ontology matching. In this configuration, all tests were executed in the same environment but an organiser had to run the tests.

The benefits of automation, as observed in SEALS, are as follows:

- Automatic and uniform results are provided.
- Since systems are not run by their designers, automation evaluates how systems are portable.
- All systems are run in the same controlled environment. As a by-product, this allows for meaningful performance measurements (see Sect. 9.3.4).
- New tests can be run on existing systems in order to check that they are correctly designed.
- Reproducibility is enhanced (results are automatically archived).

As of now, evaluation organisers still have to run evaluation campaigns, but one could imagine that system developers could run themselves tests. This would provide additional benefits: direct feedback to developers and no need for organisers to supervise evaluation synchronously.

9.2 Data Sets for Evaluation

One very important aspect of evaluation is the data set used for performing it. Finding a suitable data set is a critical issue because of the differences in form and quality of the possible data sets. We present first different factors that can influence evaluation (Sect. 9.2.1) and then discuss various data set categories (Sect. 9.2.2).

9.2.1 Dimensions and Variability of Alignment Evaluation

(Giunchiglia et al. 2009) proposed the following criteria for designing or selecting data sets for ontology matching evaluation:

⁴<http://www.seals-project.eu>.

- Complexity, i.e., that the data set is hard for state-of-the-art matching systems.
- Discrimination ability, i.e., that the data set can discriminate sufficiently among various matching approaches.
- Incrementality, i.e., that the data set allows for incrementally discovering the weaknesses of the tested systems.
- Monotonicity, i.e., that the matching quality measures calculated on subsets of gradually increasing size converge to the values obtained on the whole data set.
- Correctness, i.e., that a reference alignment is available for the data set, and allows for identifying correct and incorrect generated correspondences.

Each of the elements featured in the matching process definition (Chap. 2) can have specific characteristics that influence the difficulty of the matching task. It is thus necessary to identify and control these characteristics. We called them dimensions because they define a space of possible tests.

Characterising the variability of matching tasks helps in assessing the limitations of benchmark suites and designing benchmarks spanning the whole spectrum of matching. Indeed, for each point in this variability space a specific test could be designed. However, there could be too many of them and it is thus necessary, for each data set, to choose among the most representative values for most of these possible parameters.

These dimensions and the questions they raise are a refinement of the requirements that have been studied in Sect. 1.8. These requirements only considered general categories called *input* and *process*. Such categories need to be refined for precisely defining what a particular application can expect from a matching system, while the initial requirements concern application classes. Knowing the relations between a data set and the dimensions can be used by the application designer for finding a suitable data set with which to evaluate systems.

We review below the dimensions and justify some choices in designing benchmarks. This extends the typology introduced in (Noy and Musen 2002a; Do et al. 2002) with regard to our definition of the matching process in Sect. 2.5.1.

Although the discussions usually conclude on a least common feature set for evaluation, which can be observed from Chap. 8, OAEI has, over the years, tested many uncommon settings, e.g., input alignments, different sets of relations, application-specific evaluations.

Input Ontologies

Input ontologies o and o' can be characterised by at least three different dimensions:

Heterogeneity of the input languages: are they described in the same knowledge representation languages?

Languages: what are the languages of the ontologies? Some examples of languages include KIF, OWL, RDFS, F-Logic, UML, SQL DDL, or XML Schema.

Number: is this an alignment between two ontologies or should it match more ontologies?

As mentioned in Chap. 2, we consider here matching between homogeneous languages. The language used should be adapted to the kinds of features to be assessed by the evaluation. Thus, for example, a data set about directory matching should not be expressed in UML. However, the choice of language will also determine the systems that can be evaluated. It is admissible that not all the evaluation campaigns use the same languages.

Tasks involving multiple matching are very specific at the moment and only a small number of algorithms are considering them (He and Chang 2006; Su et al. 2006). Therefore, we consider here only evaluation of the matching results between two ontologies.

Input Alignment

The input alignment A can have the following characteristics:

Complete or update: Is the matching process required to complete an existing input alignment or is it authorised to change it?

Multiplicity: How many entities of one ontology can correspond to one entity of the other ontology?

It is reasonable to start with tests without input alignment, especially since this helps focus on the intrinsic capabilities of matchers instead of capabilities of matchers helped by input.

Parameters and Resources

Parameters p and resources r of the matching process are identified as:

Oracles and resources: Are oracles permitted? If so, which ones (the answer can be ‘any resource’)? Is user input allowed?

Training: Can training be performed on a sample?

Proper parameters: Are some parameters necessary? If so, what are they? This point is quite important when a method is very sensitive to the variation of parameters. A good tuning of these must be available.

Many systems take advantage of some external resources, such as WordNet, sets of morphological rules or a previous alignment of general-purpose catalogues, e.g., Yahoo and Google. In OAEI, for instance, it is perfectly possible to use these resources as long as they have not been specifically tuned for the evaluation. Using a sublexicon which is dedicated to the domain considered by the tests is prohibited. On the contrary, it is perfectly accepted that the algorithms prune or adapt these resources to the actual ontologies, as long as it is the normal process of the algorithm. However, this processing time must be considered within the running time.

Some algorithms may take advantage of the web for selecting resources that are adapted to the considered ontology. This is acceptable, but may compromise the replicability of the evaluation results.

In the current state, there is no consensus or valuable method for handling and evaluating the contribution of user input to the matching process, so allowing it is hard to account for.

Training on some sample is very often used by methods for matching ontologies. Thus, providing a training set can be useful for comparing algorithms based on machine learning. The training set can also be considered as some partial input alignment.

Of course, some parameters can be provided to the methods participating in the evaluation. However, these parameters must be the same for all tests. Only automatic tuning (Sect. 7.6), as part of the matching process, is acceptable.

Output Alignment

We identify the following possible constraints on the output alignment A' of the algorithm:

Multiplicity: How many entities of one ontology can correspond to one entity of the others, e.g., injective, total, one-to-one? (see Sect. 2.5.2).

Justification: Should a justification of the results be provided?

Relations: Should the relations involved in the correspondences be only equivalence relations ($=$) or could they be of other types, such as subsumption (\leq) or incompatibility (\perp).

Strictness: Can the result be expressed with degrees of confidence different from \top and \perp or should they be hardened before?

In real life, there is no reason why two independently developed ontologies should have a particular alignment multiplicity other than $*:*$. This should be the default (non) constraint on the output alignment. However, if all tests provide some particular type of alignment, e.g., $??:$ in the EON ontology tests, this introduces a bias. This bias can be suppressed by having each type of alignment equally represented. This is neither easy to find nor realistic. What would be realistic is to have a statistical evaluation of the proportion of each type of alignment.

Another worthwhile feature for users is the availability of meaningful explanations or justifications of the correspondences. However, in the absence of a standard for explanations (see Sect. 11.3) it is not possible to evaluate them at the moment.

As mentioned in Chap. 8, some systems associate a relation between entities that is different from equivalence, e.g., specificity, and some of them associate a degree of confidence to the correspondence. Concerning the relation, not all algorithms deliver the same structure, however, they can deliver equivalence. Thus, this is a common ground for evaluation. When a larger set of relations become standard, it will be worth taking them into account. As far as the degree of confidence is concerned, reference alignments should express correspondences that hold or not. It is

natural that reference alignments contain only \top confidence measure. For the resulting alignment, it is appropriate that an algorithm delivers a weighted alignment. In particular, some useful measures take these weights into account (Sect. 9.3.1). However, the lack of consensus on the interpretation of these weights renders such alignments difficult to evaluate.

Matching Process

The matching process f itself can be constrained by:

Resource constraints: Is there a maximal amount of time or space available for computing the alignment?

Language restrictions: Is the matching scope limited to some kind of entities, e.g., only classes?

Property: Must some property be true of the alignment? For instance, one might want that the alignment is satisfiable (as defined in Chap. 2) or that it preserves consequences or that the initial alignment is preserved, i.e., $o, o', A' \models A$.

Resource constraints can be considered either as constraining the amount of a resource or a measure of the amount consumed (Sect. 9.3.4). It is an important factor, at least for comparative evaluation, and must be measured. It can also be measured for competence benchmarks, even if it is difficult due to the heterogeneity of the environments in which benchmarking is performed.

Constraints on the kind of language construct to be found in alignments can be designed. However, currently very few matching algorithms can match complex expressions. Most of them match the identified (named) entities and some of them are only restricted to concepts. With regard to its importance and its coverage by current matching systems, it makes sense to ask for matching named entities and consider complex expressions later.

The properties of the alignments provided by the matching algorithms are not very often mentioned and they are very heterogeneous depending of the implemented techniques. It is thus difficult to ask for particular properties.

9.2.2 Examples of Data Sets

Datasets for matching ontologies are not easy to find. The first problem is that they require pairs of public and well-designed ontologies with a meaningful overlap.

Moreover, for evaluating the matching algorithms they should also provide reference alignments, making them even scarcer.

In addition, it is necessary to take into account the quality of the ontologies and alignments: the ontologies are more interesting if their matching reflects realistic matching problems and the alignments must be correct or, at least, be the expected ones.

Most of the reference alignments used for evaluation involve human judgements. However, humans are not usually very good at matching ontologies manually (Tordai et al. 2011). Thus, it would be useful to evaluate the task in which alignments are embedded, e.g., information retrieval, web service invocation, instead of matching itself.

Below are some examples of data sets that have been used so far.

OAEI Systematic Benchmark Suite

The data set made for the first OAEI campaign is an artificial data set built from one OWL ontology on the bibliography topic. It contains 33 named classes, 24 object properties, 40 data properties, 56 named individuals and 20 anonymous individuals.

This initial ontology is systematically and automatically altered by distorting the features of ontology languages, e.g., names, properties, subclass relations. The alteration results in a set of more than 50 pairs of ontologies. The kind of proposed alignments is still limited: they only match named classes and properties and they mostly use the equivalence relation with confidence of 1.

This data set can be considered as correct by construction. It is not realistic nor very hard: it is based on small tests and offers some easy ways to reach the correct result. It is especially made for evaluating the strengths and weaknesses of matching systems. The initial data set has been used for every OAEI campaign in order to measure the evolution of the field.

Since 2011, OAEI uses a benchmark suite generator able to generate similar test suites from any ontology (see Sect. 9.2.3).

Large Scale Ontology Sets

There is a need for large scale ontologies to be matched. One early attempt to do this has been reported in (Zhang et al. 2004). It consists in matching two large ontologies from the domain of anatomy. The two considered ontologies are FMA and Galen.

These are huge real-world ontologies which contain classes, relations, text documentation and labelling. These ontologies contain thousands of classes and barely no instance data.

This test case is obviously realistic. The experiments reported in (Zhang et al. 2004) show that it is difficult, and helps in finding weaknesses. The main problem with the task is that it does not have a widely acknowledged reference alignment.

In the same spirit, OAEI 2012 proposed the task of matching FMA, SNOMED CT, and the (US) National Cancer Institute Thesauri. The full ontologies or thesauri contain respectively 78 989, 306 591, and 66 724 classes. The reference alignments are taken from UMLS (Sect. 2.1.2) and come in two versions: the raw and inconsistent version and a refined consistent version. They respectively contain around 3 000 (FMA-NCI), 8 000 (FMA-SNOMED) and 18 000 (SNOMED-NCI) correspondences. The test sets are further decomposed into three sets of increasing size.

Directory Sets

(Avesani et al. 2005) proposed a practical way to build test sets for matching web directories or classifications. This has the advantage of generating automatically a test set that changes with time. The idea is to consider two web directories, i.e., a hierarchy of topical web pages that index web pages, and to take advantage of the corresponding indexed pages for deciding if two topics are equivalent. This can already be considered as a matching technique in itself (Sect. 8.2.8).

This technique has been used in OAEI campaigns between 2005 and 2010. The collection reported in (Avesani et al. 2005) contained from 300 000 to 800 000 topics. It has been characterised by the authors as a difficult test.

Thesauri

There is currently a wealth of resources available from the fields of libraries, museums and more generally cultural heritage. These are thesauri covering hierarchies of concepts considered as terms and large amounts of textual knowledge, usually about pieces of art. A huge interest in using different thesauri together has created the need for matching them.

For OAEI 2006, a pair of thesauri on the topic of food has been provided: AGROVOC is a thesaurus for the Food and Agriculture Organisation and NAL is a thesaurus from the US Agricultural department. They respectively contain 16 000 and 41 000 terms. The two test sets have been made available in SKOS (Sect. 10.1.7).

This task can be considered as a representative of real-world matching problems. These thesauri are certainly challenging with respect to their size. However, at the moment, there is only a partial reference alignment for these tests.

Other Test Collections

The Illinois Semantic Integration Archive had offered until recently some of the data sets used for evaluating various schema matchers presented in Chap. 8. Some data sets for schema matching are also available: THALIA (Hammer et al. 2005) is a data set of XML university course descriptions which is used for testing data integration through queries (in XQuery), and (Marie and Gal 2008) offers a data set on finance used by schema matchers. OAEI also uses the OntoFarm data set made of ontologies on the conference domain coming from various origins. This data set is assumed to be realistic and has the advantage of providing several ontologies of the same domain. The MultiFarm data set (Meilicke et al. 2012) is a translation of OntoFarm in 7 different languages (see Table 9.2). In addition, the ontology matching web page refers to test cases that have been built and published by matching tools designers.⁵

⁵<http://www.ontologymatching.org/evaluation.html>.

Table 9.2 summarises the characteristics of several available data sets used in OAEI.

Table 9.2 Characteristics of test cases (adapted from (Euzenat et al. 2011)). ‘open’ evaluation is made with already published reference alignments, ‘blind’ evaluation is made by organisers from reference alignments unknown to the participants and ‘expert’ evaluation involves post hoc analysis of results, by expert users.

Dataset	Formalism	Relations	Confidence	Modalities	Language
benchmarks	OWL	=	[0 1]	open	EN
generated b.	OWL	=	[0 1]	blind	EN
anatomy	OWL	=	[0 1]	blind	EN
largebio	OWL	=, ≤, ≥	[0 1]	open	EN
conference	OWL-DL	=, ≤	[0 1]	blind + open	EN
multifarm	OWL-DL	=, ≤	[0 1]	blind + open	EN + CN + RU + FR + DE + NL + PT
directory	OWL	=, ≤, ≥, ⊥	1	blind + open	EN
library	SKOS + OWL	exact-, narrow-, broadMatch	1	blind	EN + NL + FR
benchsubs	OWL	=, ≤, ≥	[0 1]	open	EN
ars	RDF	=	[0 1]	open	EN
tap	RDF	=	[0 1]	open	EN
iimb	RDF	=	[0 1]	open	EN
vlcr	SKOS + OWL	exact-, closeMatch	[0 1]	blind expert	NL + EN

9.2.3 Test Generation

Instead of using existing data sets, it may be worth generating test cases automatically. This has several advantages, such as (i) always providing a new test set, instead of reusing known ones, (ii) controlling the parameters of the test set to what has to be tested, and (iii) generating tests of various sizes for scalability experiments.

The main principle behind test set generation is that the generator introduces variability, i.e., the generated sets are different, yet they can be controlled so that a matcher always obtains results whose evaluation is the same, for a test generated with the same parameters. This may require averaging over several tests instead of a single test.

The OAEI benchmark generator starts with one ontology and generates a modified ontology and the alignment between the two ontologies. Because the program generates the target ontology from the source ontologies, it records, without contestation, in the alignment the correspondences between entities. This generation is

based on elementary transformations (alterators) which apply one type of transformation. Alterators may be applied (i) partially so as to affect only a portion of the ontologies, and (ii) sequentially to generate test cases with different transformations.

Several efforts took inspiration from the original OAEI Benchmark for generating test sets through alteration, e.g., for generating test sets for geographic web services (Vaccari et al. 2012). In general, such efforts tend to publish the resulting test set and not the generator. For example, both IIMB (Ferrara et al. 2008) and ONTOBI (Zaiss et al. 2010) benchmarks have been developed for testing instance-based ontology matching systems. On the same basis as the OAEI benchmark suite, they add a large number of instances based on external sources (the Internet Movie Database or Wikipedia data). The Swing approach (Ferrara et al. 2011a) is a further effort to generate instance matching test sets taking the same approach. It introduces a new interesting way of selecting data sets from which to generate benchmarks by using patterns, and offers a variety of ways for altering data on the basis of an intermediate ontology.

In XML schema matching, SGen (the schema generation part of STBenchmark) (Alexe et al. 2008c, 2008b) generates tests (pairs of schemas) by altering a source schema based on the specification of 11 scenarios (base alterators). The scenarios are defined through a set of input parameters which include the characteristic parameter, e.g., nesting depth, number of subelements, length of joint paths, the standard deviations to be applied for sampling each characteristic, and a repetition parameter. It does not generate a reference alignment because it tests the result of the transformation from source instances into target instances.

The benchmarks of database schema matching systems closest to OAEI are those of X BenchMatch (Duchateau et al. 2007a). Schema instances can also be included in the generated schemas. X BenchMatch (Duchateau et al. 2007a) provides reference integrated schema for evaluating matchers. It can be extended programmatically.

The initial OAEI Benchmark was automatically generated, but in an ad hoc way: the generator applied to only one ontology. The Alignment API (Sect. 10.3.6) features a test generator API (Euzenat et al. 2013). It offers an extendible library of alterators which may be used programmatically for generating different test sets from different seed ontologies and different alteration modalities. It allows different modifications at each run of the program and the set of input parameters may be adjusted in order to cover the problem space with any precision. It has been used for reproducing the OAEI benchmark suite both with the original seed ontology and with other ontologies. The generator has been used for generating test sets on the model of OAEI benchmarks with different seed ontologies and different modalities. This largely improved the variability of generated tests. Some of these generated tests have been used in OAEI since 2011.

An automatic generator aiming at following a precise distribution has been proposed in (Tournaire 2010). This approach generates the structure and the instances of two taxonomies, and an alignment between these two generated taxonomies. Both taxonomies must have a fixed size, and a Boltzmann sampler is used to achieve this. The probabilistic model used ensures an equal probability of appearance of a tree having a given size. Therefore, the input data is controlled using this sampler.

9.3 Evaluation Measures

In order to evaluate the results of matching algorithms, it is necessary to confront them with ontologies to be matched and to compare the alignments produced with a reference alignment based on some criteria.

This section considers different possible measures for evaluating matching algorithms and systems. This includes both qualitative and quantitative measures. We divide them into compliance measures and performance measures. Compliance measures evaluate the degree of conformance of returned alignments to what is expected. We will present some classical evaluation measures (Sect. 9.3.1) and measures especially designed for ontology matching evaluation (Sect. 9.3.2). We also discuss measures designed to overcome the lack of a full reference alignment (Sect. 9.3.3). Performance measures account for such features of algorithms as speed and memory consumption (Sect. 9.3.4). User-related measures focus on evaluation of user interaction with a matching system (Sect. 9.3.5).

9.3.1 Compliance Measures

Compliance measures evaluate the degree of compliance of a system with regard to some standard. One common standard is the use of a reference alignment R to which the result from the evaluated matching algorithm A is compared. In what follows, the alignments A and R are considered to be sets of correspondences, being pairs of entities. As noted before (Sect. 9.2), such a reference output is not always available, not always useful and not always consensual. However, for the purpose of benchmarking, we can assume that it is desirable to provide such a reference.

Let us consider the case of the evaluation of three alignments on classes that must be compared to the alignment of Fig. 2.9 (p. 45). This alignment is considered to be the reference alignment (R) and contains three correspondences:

$$\text{Book} = \text{Volume} \quad \text{Person} = \text{Human} \quad \text{Science} = \text{Essay} \quad (R)$$

The alignments to be compared are nearmiss (A_1), farone (A_2) and noncomplete (A_3). These are presented in Fig. 9.2. They are made of the following correspondences:

$\text{Product} = .4 \text{ Volume}$	$\text{Person} = .7 \text{ Writer}$	$\text{Science} = .9 \text{ Essay}$	(A_1)
$\text{Book} = .8 \text{ Volume}$	$\text{Children} = .4 \text{ Literature}$	$\text{Pocket} = .8 \text{ Essay}$	(A_2)
$\text{Book} \leq .7 \text{ Volume}$	$\text{Person} \geq .7 \text{ Writer}$	$\text{Person} \leq .6 \text{ Human}$	(A_3)

A first simple distance between two sets is the Hamming distance. It measures the dissimilarity between two alignments by counting the joint correspondences with regard to the correspondences of both sets.

Definition 9.1 (Hamming distance) Given a reference alignment R , the Hamming distance between R and some alignment A is a dissimilarity $H : \Lambda \times \Lambda \rightarrow [0, 1]$ defined as

$$H(A, R) = 1 - \frac{|A \cap R|}{|A \cup R|}.$$

Example 9.2 (Hamming distance on alignments) Taking the class part of the alignment of Fig. 2.9, as the reference alignment R , we can compare it with the result given by the alignments of Fig. 9.2. The shorter the distance, the better. For A_1 and A_2 , the Hamming distance between these alignments is very high: .8. Indeed, both alignments only found one correct correspondence out of three. Thus, this results in two inaccurate correspondences and two missed correspondences. The distance is even higher for A_3 (1.).

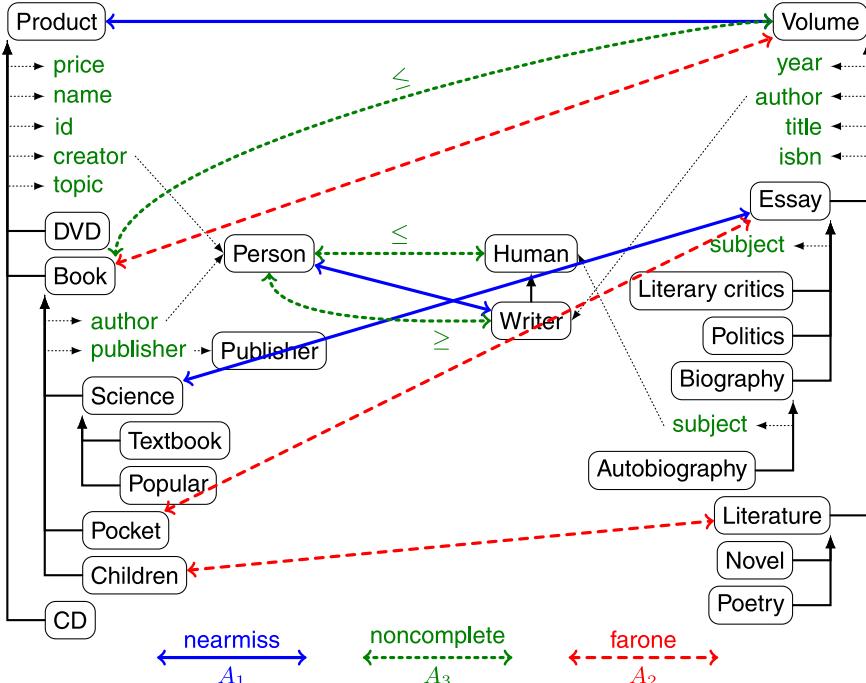


Fig. 9.2 Three class alignments between the ontologies of Fig. 2.7. These are to be compared with the alignment of Fig. 2.9 restricted to class correspondences.

The most prominent criteria are *precision* and *recall* originating from information retrieval (van Rijsbergen 1975) and adapted to ontology matching (Do et al. 2002). Precision and recall are based on the comparison of the resulting alignment

A with a reference alignment R , effectively comparing which correspondences are discovered and which are not. These criteria are well understood and widely accepted.

Precision measures the ratio of correctly found correspondences (true positives) over the total number of returned correspondences (true positives and false positives; see Fig. 9.3). In logical terms, precision is meant to measure the degree of correctness of the method.

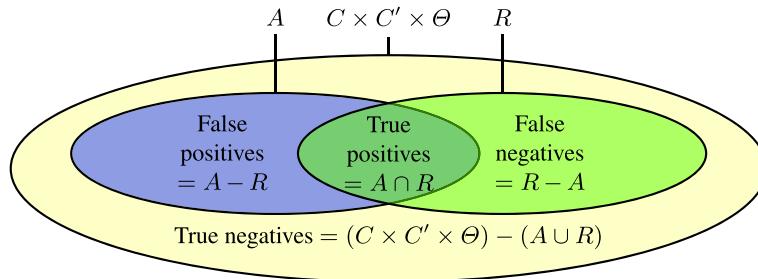


Fig. 9.3 Two alignments as sets of correspondences and relations between them.

Definition 9.3 (Precision) Given a reference alignment R , the precision of some alignment A is a function $P : \Lambda \times \Lambda \rightarrow [0, 1]$ such that

$$P(A, R) = \frac{|R \cap A|}{|A|}.$$

Precision can also be determined without explicitly having a complete reference alignment. In this case, only the correct correspondences among the retrieved correspondences have to be determined, namely $R \cap A$.

Recall measures the ratio of correctly found correspondences (true positives) over the total number of expected correspondences (true positives and false negatives). In logical terms, recall is meant to measure the degree of completeness of the alignment.

Definition 9.4 (Recall) Given a reference alignment R , the recall of some alignment A is a function $R : \Lambda \times \Lambda \rightarrow [0, 1]$ such that

$$R(A, R) = \frac{|R \cap A|}{|R|}.$$

In the definition above, the letter R stands for both the recall function and the reference alignment. Since one is a function and the other is a set, these are easy to be distinguished by their use even if referred by the same letter.

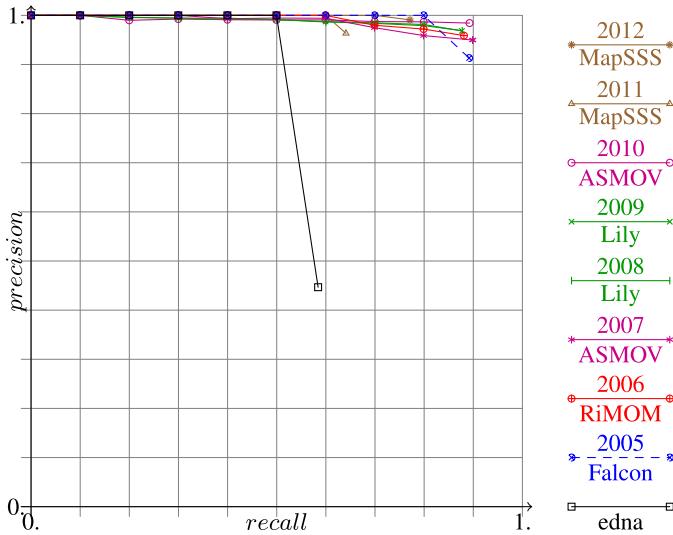


Fig. 9.4 Precision/recall curves for the yearly best results on the OAEI benchmark data set 2007–2010 (edna, edit distance on class names, is a simple matcher used as baseline). The curves are obtained from the provided alignments in which correspondences are ordered by decreasing confidence. For each decimal value v between 0 and 100, the algorithm selects the first correspondences in order to reach $v\%$ recall, i.e., the corresponding percentage of correspondences from the reference alignment, and it reports the precision at that point, i.e., with only these correspondences.

When comparing systems in which precision and recall can be continuously determined, it is convenient to draw the precision/recall curve, or precision@ k , i.e., precision at $k\%$ recall, and compare these curves (see Fig. 9.4). There are two advantages to these curves: (i) they allow for the comparison of alignments with confidence measures; (ii) they are independent from the interpretation of confidence: only its induced order is relevant. More generally, it is always possible to plot any measure@ k to understand what recall the optimal measure would provide. This kind of measure is widespread when presenting results (in the TREC competitions for instance). It is computed slightly differently here because no interpolation is produced.

Noise and silence are the complement measures of precision and recall. These are defined as follows: $\text{Noise}(A, R) = 1 - P(A, R)$ and $\text{Silence}(A, R) = 1 - R(A, R)$.

The fallout measures the percentage of false positives over the incorrect correspondences, and the miss measures the percentage of false negatives over the non-retrieved correspondences.

Definition 9.5 (Fallout) Given a reference alignment R , the fallout of some alignment A is a function $F : \Lambda \times \Lambda \rightarrow [0, 1]$ such that

$$F(A, R) = \frac{|A \setminus R|}{|C \times C' \times \Theta \setminus R|}.$$

Definition 9.6 (Miss) Given a reference alignment R , the miss of some alignment A is a function $M : \Lambda \times \Lambda \rightarrow [0, 1]$ such that

$$M(A, R) = \frac{|R \setminus A|}{|C \times C' \times \Theta \setminus A|}.$$

Although precision and recall are the most widely and commonly used measures, when comparing systems one may prefer to have only a unique measure. Moreover, systems are often not comparable based solely on precision or recall. The one which has higher recall may have a lower precision and vice versa. So, it is not a good idea to compare systems on precision or recall alone. For this purpose, two measures are introduced which aggregate precision and recall: F-measure and overall.

Definition 9.7 (F-measure) Given a reference alignment R and a number α between 0 and 1, the F-measure of some alignment A is a function $M_\alpha : \Lambda \times \Lambda \rightarrow [0, 1]$ such that

$$M_\alpha(A, R) = \frac{P(A, R) \times R(A, R)}{(1 - \alpha) \times P(A, R) + \alpha \times R(A, R)}.$$

If $\alpha = 1$, then the F-measure is equal to precision and if $\alpha = 0$, the F-measure is equal to recall. In between, the higher the value of α , the more importance is given to precision with regard to recall. Very often, the value $\alpha = 0.5$ is used, i.e.,

$$M_{0.5}(A, R) = \frac{2 \times P(A, R) \times R(A, R)}{P(A, R) + R(A, R)}.$$

This is the harmonic mean of precision and recall.

The F-measure attempts at balancing the importance of precision and recall so that it is not possible to compensate one for the other: if they are equal, then the F-measure will be the same, but if they are not equal, F-measure will be below their average:

$$\min(P(A, B), R(A, B)) \leq M_\alpha(A, B) \leq \frac{P(A, B) + R(A, B)}{2}$$

Such a measure may be used for selecting the parameters, in particular, a threshold to put on the results, such that the F-measure is optimal. Moreover, it allows comparing systems by their precision and recall at the point where their F-measure is maximal.

The overall measure, also defined in (Melnik et al. 2002) as matching accuracy, is the ratio of the number of errors on the size of the expected alignment. It may be considered as an edit distance between an alignment and a reference alignment in which the only operation is ‘error correction’. In this respect, it is considered as a measure of the effort required to fix the alignment. The overall is always lower than the F-measure and it ranges between $[-1, 1]$. In fact, if precision is under .5 overall has a negative value denoting that repairing the alignment is not worth the effort.

Definition 9.8 (Overall) Given a reference alignment R , the overall of some alignment A is a function $O : \Lambda \times \Lambda \rightarrow [-1 1]$ such that

$$O(A, R) = R(A, R) \times \left(2 - \frac{1}{P(A, R)}\right).$$

Alternatively, it can also be defined as

$$O(A, R) = 1 - \frac{|(A \cup R) - (A \cap R)|}{|R|} = 1 - \frac{|R - A| + |A - R|}{|R|}.$$

Table 9.3 illustrates the use of these measures.

Table 9.3 Results of the evaluation measures for the three alignments of Fig. 9.2 as well as for the reference alignment itself. For Fallout and Miss, values are expressed with respect to ϵ which is in this case around 10^{-3} .

Alignment	Precision	Recall	F-measure	Fallout	Miss	Overall
refalign	1.00	1.00	1.00	1.00	0.00	0.00
nearmiss (A_1)	0.33	0.33	0.33	2ϵ	2ϵ	-0.33
farone (A_2)	0.33	0.33	0.33	2ϵ	2ϵ	-0.33
noncomplete (A_3)	0.00	0.00	0.00	3ϵ	3ϵ	0.00

Typically, when using the same matcher on different tasks, performances may vary. One strategy with respect to this is to test matchers on tasks similar to a target task (see Sect. 9.4). Another strategy is to evaluate the measure on various different tasks and try to deduce guarantees on any new tasks. (Niu et al. 2011) proposed measures of stability of a matcher that aggregate the results of several evaluations. Roughly speaking, these measures compute for each task the F-measure optimal threshold, i.e., the threshold for which, if the alignments are reduced to correspondences above it, F-measure is optimised. The smaller the standard deviation of this measure or the smaller the standard deviation of the difference between the optimal F-measure and the F-measure obtained by the average F-measure optimal threshold, the more stable the matcher, i.e., the closest to the optimum the average of these values is. This method does not only provide a stability evaluation measure, but an effective means of computing the threshold.

9.3.2 Generalising Precision and Recall

The classical precision and recall measures described above consider the alignments as simple sets. This means that they do not take into account specifics of alignments such as:

- They may be assigned confidence: an incorrect correspondence with low confidence is better than one with high confidence;
- They may be interpreted in the context of the ontology structure: a correspondence which is close to the expected one is better than one which is remote;
- They may be interpreted logically: a correspondence which is entailed by the reference alignment should be considered correct.

This does not help in comparing alignments: the lines A_1 and A_2 of Table 9.3 are identical.

As precision and recall are well-understood measures, it is preferable to extend them. This also ensures that measures derived from precision and recall, e.g., F-measure, still can be computed easily. We provide below three generalisations of precision and recall dealing with these problems.

Weighted Precision and Recall

Confidence-weighted precision and recall are designed for taking into account confidence provided by matchers in the evaluation of precision and recall. They assess the capacity of matchers to provide good confidence in a complementary way as precision and recall. For that purpose, if a matcher has provided an incorrect correspondence, but with a low confidence, it should not be penalised by 1, but by a measure of its confidence. Similarly, if it has provided a correct correspondence with a low confidence, it should not be rewarded more than its confidence. Weighted precision and recall are defined as follows.

Definition 9.9 (Weighted precision and recall) Given a reference alignment R , the weighted precision of some alignment A is given by

$$P_W(A, R) = \frac{\sum_{c \in A} 1 - |\kappa_A(c) - \kappa_R(c)|}{|A|}$$

and weighted recall is given by

$$R_W(A, R) = \frac{\sum_{c \in R} 1 - |\kappa_A(c) - \kappa_R(c)|}{|R|}$$

Confidence values (given by the κ function) are assumed to have a difference operation ranging between 0. and 1. These measures take into account confidences in the reference alignment for preserving the symmetry of precision and recall. But when the reference has only correspondences with full confidence, as usual in OAEI, the measures may be simplified. However, this may also be used for dealing with reference alignment assessed by several person: in this case, each particular correspondences can be assigned a confidence corresponding to the ratio of assessors who have considered the correspondence correct:

$$\kappa_R(c) = \frac{|\text{assessors choosing } c|}{|\text{assessors}|}$$

These measures are useful because they integrate the capacity of the matcher to highlight the correspondences worthy of attention by a human user. Contrary to precision/recall curves, however, they do not deal with the different scales of confidence that matchers may use.

If an alignment contains only 1. confidences, then it scores exactly as in classical precision and recall (given that the reference contains only 1. as well). Otherwise, it scores for the amount of confidence. If the correspondence is correct, this will contribute to decreasing recall—it will be counted for less than 1.—, if the correspondence is incorrect, this will increase precision—by counting the mistake for less than 1. This rewards systems able to provide accurate confidence measures (or penalises less mistakes on correspondences with low confidence). So, weighted recall may decrease if correct correspondences are given with low confidence (they may also decrease precision in case the reference alignment has lower confidence).

Relaxed Precision and Recall

Classical precision and recall have a drawback: whatever correspondences have not been discovered are definitely not considered (all-or-nothing). As a result, they do not discriminate between an alignment that may be very close to the expected result and another remote from it, and they do not measure the effort required from users to correct alignments. In fact, the alignment A_1 of Fig. 9.2 is arguably better than the alignment A_2 . However, as testified in Table 9.3, they score exactly the same for all presented measures.

Often, it makes sense to not only have a decision whether a particular correspondence has been found or not, but also a measure of the proximity of the found alignments. This implies that near misses are also taken into consideration instead of only exact matches.

Moreover, the alignments have to go through users scrutiny and correction before being used. Therefore, it is worth measuring the effort required from users for correcting the provided alignment instead of only if some correction is needed. This also calls for a relaxation of precision and recall.

Similar remarks have been made in computational linguistics and some solutions have been proposed in (Langlais et al. 1998; Sun and Lin 2001). In the context of alignment evaluation, (Ehrig and Euzenat 2005) investigated relaxing precision and recall in order to overcome these problems. Relaxing precision and recall amounts to measuring the proximity of alignments rather than the strict size of their overlap. Instead of taking the cardinality of the intersection of the two sets $|R \cap A|$, natural relaxations of precision and recall measure their proximity $\omega(A, R)$. Moreover, the relaxed measure is required to preserve positiveness, maximality and boundedness satisfied by $|R \cap A|$.

Definition 9.10 (Relaxed precision and recall) Given a reference alignment R and an overlap function ω between alignments, the relaxed precision of an alignment A

is

$$P_\omega(A, R) = \frac{\omega(A, R)}{|A|},$$

and relaxed recall is

$$R_\omega(A, R) = \frac{\omega(A, R)}{|R|},$$

such that ω satisfies the following conditions:

$$\forall A, B, \omega(A, B) \geq 0 \quad (\text{positiveness})$$

$$\forall A, B, \omega(A, B) \leq \min(|A|, |B|) \quad (\text{maximality})$$

$$\forall A, B, \omega(A, B) \geq |A \cap B| \quad (\text{boundedness})$$

Symmetry is not required, especially since R and A are not in symmetrical positions: the former is the reference and the latter is judged against it. There are many different ways to design a proximity between two sets satisfying these properties.

From this simple set of constraints, several concrete measures detailed in (Ehrig and Euzenat 2005) have been designed:

Symmetric calculates the distance in the ontologies between the found entities and the reference ones. The closer they are, the higher the similarity.

Effort-based computes the effort necessary to modify the errors found in the alignments: it is based on a model of what is involved in modifying an alignment through an alignment editor for retrieving the reference alignment. This measure is arguably better than the overall (Melnik et al. 2002) presented before because it can weight differently different errors depending on the difficulty to correct it which itself depends on the editing environment used.

Oriented is a specific measure which uses a different ω for precision and recall depending on the impact an error has on these measures. For example, when one wants to retrieve instances of some class, a subclass of the expected one is correct but not complete and thus affects recall but not precision. This measure is targeting application-specific evaluation (see Sect. 9.4).

If these proposed precision and recall measures are applied to the alignments of Fig. 9.2, they yield the results of Table 9.4. They mainly illustrate entity pair similarities, as relations and confidences are always identical. For the oriented measure we assume that the query is given in ontology o and the answer has to be retrieved from ontology o' . Since the oriented measure is nonsymmetric, one has to define the direction beforehand.

The presented relaxed measures keep precision and recall untouched for the best alignment R and they help discriminate between irrelevant alignments, such as A_2 (farone), and those which are not far from target ones, like A_1 (nearmiss). However, they do not, except partially the oriented measure, account for the correctness of A_3 (noncomplete).

Semantic Precision and Recall

For solving this last problem, semantic versions of precision and recall have been proposed (Euzenat 2007). The natural semantic extension of these measures consists of using the set of α -consequences (see Sect. 2.5.3) instead of $|A \cap R|$. So, A and R are replaced by their respective α -closures: $Cn_{\{o, o'\}, \{A\}}^\alpha(o, o')$ and $Cn_{\{o, o'\}, \{R\}}^\alpha(o, o')$ (abbreviated into $Cn^\alpha(A)$ and $Cn^\alpha(R)$). In this case, true positives are the correspondences that are consequences of both alignments, and the usual definitions of true and false positives and negatives are extended to alignment consequences.

Definition 9.11 (Ideal semantic precision and recall) Given a reference alignment R , the ideal semantic precision of some alignment A is

$$P_{ideal}(A, R) = \frac{|Cn^\alpha(R) \cap Cn^\alpha(A)|}{|Cn^\alpha(A)|}$$

and the ideal semantic recall is

$$R_{ideal}(A, R) = \frac{|Cn^\alpha(R) \cap Cn^\alpha(A)|}{|Cn^\alpha(R)|}$$

This ideal way of dealing with semantic precision and recall can be applied to any language with a semantics. It is not restricted to alignments and as soon as the notion of consequence is defined, it can be applied. However, it has a major drawback: both numerator and divisor may be infinite, yielding an undefined result. Even with finite ontologies, this can come from the expressiveness of the entity language. If $c \leq (\text{atmost } np')$, then for any $n' \geq n$, $c \leq (\text{atmost } n' p')$ is an α -consequence.

So, semantic precision and recall were proposed, which avoid this problem:

Definition 9.12 (Semantic precision and recall) Given a reference alignment R , the semantic precision of some alignment A is

$$P_{sem}(A, R) = \frac{|A \cap Cn^\alpha(R)|}{|A|}$$

and the semantic recall is

$$R_{sem}(A, R) = \frac{|Cn^\alpha(A) \cap R|}{|R|}$$

Both values are defined when the alignments are finite. Moreover, the measures can be computed if there exists a complete and correct prover for the languages because there is always a finite set of assertions to check, i.e., $Cn^\alpha(A) \cap R = \{\mu \in R; A \models \mu\}$. It is still possible to artificially improve precision of this last version, by adding all consequences to A , so adaptations have been provided (David and Euzenat 2008). This measure has been implemented in (Fleischhacker and Stuckschmidt 2010) and in the Alignment API (Sect. 10.3.6).

Examples of measures introduced so far are given in Table 9.4. From Table 9.4, the precision of Alignment A_3 has become 1. showing the correctness of the alignment (all its correspondences are entailed by the reference alignment).

Table 9.4 Generalised precision and recall results on the alignments of Fig. 9.2 (best results are in bold).

ω	reference		nearmiss		farone		noncomplete	
	(R, R)		(R, A_1)		(R, A_2)		(R, A_3)	
	P	R	P	R	P	R	P	R
standard	1.00	1.00	0.33	0.33	0.33	0.33	0.00	0.00
weighted	1.00	1.00	0.30	0.30	0.27	0.27	0.00	0.00
symmetric	1.00	1.00	0.50	0.50	0.33	0.33	0.33	0.33
effort-based	1.00	1.00	0.53	0.53	0.33	0.33	0.33	0.33
oriented	1.00	1.00	0.66	0.50	0.33	0.33	0.50	0.50
semantic	1.00	1.00	0.33	0.33	0.33	0.33	1.00	0.00

9.3.3 Sampling and Relative Precision and Recall

It may happen that a reference alignment is not available, in particular because the population to consider, i.e., $Q_L(o) \times Q_{L'}(o') \times \Theta$, is far too large to assess precisely. In this case, a reference alignment is usually built on a subset A^0 of this population.

In order to do this, one must first select a sample on which to compute the measures (this is called *sampling*). Then this sample has to be evaluated by assessors.

There are typically three techniques for sampling:

- simple sampling selects a fully random subset of the population,
- stratified sampling considers that the population is divided into subsets and random sampling is applied to each of these subsets (measures can then be weighted by the respective sizes of these subsets),
- clustered sampling considers that the population is clustered into subpopulations, each representative of the whole population, and then several such clusters are selected.

In random sampling, the probability of obtaining correct correspondences are usually low and this requires to select a still large space. In ontology matching, sampling may also be achieved by first sampling the ontologies and taking the whole correspondence space between ontology samples. (van Hage 2007) studied the various methods of sampling and assessing alignments and proposed using stratified sampling for precision and clustered sampling for recall.

An alternative strategy, called *pooling*, consists of selecting samples from among the correspondences returned by the matchers to evaluate. A sample is then obtained and used as a subset of the correspondence space. It may be obtained by merging together all correspondences returned by each system or a randomly selected number of correspondences from each system.

Once the sample A^0 has been determined it has to be evaluated, i.e., to be decided which correspondences are correct. This sample alignment is then evaluated by experts for correctness. This provides a partial reference alignment $R^0 \subseteq A^0$. It is advised to obtain several evaluations from several assessors. Indeed, from one person to another, the reference alignment may vary a lot (Tordai et al. 2011). It is thus useful to retain the consensus between assessors, either by voting or by reaching an agreement.

Once a reference alignment is available, it is possible to define relative precision and recall.

Definition 9.13 (Relative precision and recall) Given a partial reference alignment R^0 on a sample alignment A^0 , the relative precision of some alignment A is

$$P^0(A, R^0) = P(A \cap A^0, R^0) = \frac{|A \cap R^0|}{|A \cap A^0|}$$

and relative recall is

$$R^0(A, R^0) = R(A \cap A^0, R^0) = \frac{|A \cap R^0|}{|R^0|}$$

We call these two measures relative because they are relative to the sample that has been extracted. These are not anymore absolute measures.

Recall is expected to be higher than actual recall because it is based only on correspondences that at least one system returned, leaving aside those that no system were able to return.

Another related measure is residual recall or recall + (Euzenat et al. 2007a), which consists of determining an alignment B from a baseline algorithm and measuring recall only on correspondences not in this alignment. This measure concentrates on the hardest part of matching.

Definition 9.14 (Residual recall) Given a reference alignment R and a set of correspondences $B \subseteq R$ considered as easy, the residual recall of some alignment A is given by

$$R^B(A, R) = R(A, R \setminus B) = \frac{|A \cap R \setminus B|}{|R \setminus B|}$$

A relative version of residual recall would consist of taking the intersection of all the alignments to evaluate as B .

9.3.4 Performance Measures

Performance measures assess the resources consumed for matching two ontologies. They may be used when the algorithms are 100 % compliant or balanced against compliance (Ehrig and Staab 2004). We mention some of these criteria below.

Unlike compliance measures, performance measures depend on the processing environment and the underlying ontology management system. Thus, it is difficult to obtain objective evaluations. It is important that compared algorithms are run under the same conditions.

Speed

Speed is measured by the amount of time taken by the algorithms for performing their matching tasks. It should be measured in the same conditions, i.e., same processor, same memory consumption, for all the systems. Other measures may involve limited resources. If user interaction is required, one has to ensure that only the processing time of the matching algorithm is measured.

Network

Besides speed, some systems may use network connectivity in order to achieve matching, typically for accessing external ontologies (see Sect. 7.3) or alignments. The network consumption may be measured or limited in terms of bandwidth or throughput.

Memory

The amount of memory used for performing the matching task is another performance measure. Due to the dependency with underlying systems, it could also make sense to measure only the extra memory required in addition to that of the ontology management system, but it still remains highly dependent.

Scalability

There are two possibilities for measuring scalability, at least in terms of speed and memory requirements. Firstly, it can be assessed by a theoretical study. Secondly, it can be assessed by evaluation campaigns with quantified tests of increasing complexity. From the results, the relationship between the complexity of the test and the required amount of resources can be represented graphically and the mathematical relationship can be approximated.

9.3.5 User-Related Measures

So far, the measures have been machine focussed. In some cases, algorithms or applications require some kind of user interaction. This can range from users using the alignment results to concrete user input during the matching process. In this case, it is even more difficult to obtain an objective evaluation. Below we discuss measures which involve users in the evaluation loop.

Level of User Input Effort

If algorithms require user intervention, this intervention may be measured in terms of some elementary information users provide to the system, e.g., the number of correspondences. When comparing systems which require different input or no input from users, it is necessary to consider a standard for elementary information to be measured. This is not an easy task.

Oracle-Based Measures

It is possible to organise the evaluation of matchers requiring user input by allowing matchers to query at any time an oracle for the relation between two entities. When a reference alignment is available, the oracle can be easily simulated: the protocol would constrain matchers to providing alignments before and after oracle queries, so that it would be possible to compute measure@ k , which stands for any of the standard measures (Sect. 9.3.1) and such that k is the number of queries to the oracle (Paulheim et al. 2013).

Such a protocol would have the advantage of being independent from the use of the input for anchoring, disambiguating (Sect. 7.8.1), debugging (Sect. 7.8.2), or tuning (Sect. 7.6) matchers.

General Subjective Satisfaction

From a use case point of view it makes sense to directly measure user satisfaction. As this is a subjective measure, it cannot be assessed easily. Extensive preparations have to be made to ensure a valid evaluation. Almost all of the objective measures mentioned so far have a subjective counterpart. Possible measurements include

- input effort,
- speed,
- resource consumption, e.g., memory,
- output exactness, related to precision,
- output completeness, related to recall, and
- understandability of results, e.g., explanations.

Due to its subjective nature numerical ranges as evaluation results are less appropriate than qualitative values, such as very good, good, or satisfactory.

9.4 Application-Specific Evaluation

So far evaluation has been considered in general. However, the evaluation could also be considered in the context of a particular application or a particular kind of applications. Application-specific evaluation is dedicated to find a suitable system for a particular task. This is especially useful for application designers who need to integrate a matching system and this complements the requirement satisfaction analysis presented in Sect. 8.5.

There are two main complementary ways to design application-specific evaluations: (i) using a specific test set and experiment design; (ii) interpreting the results with an application-oriented bias. As a matter of fact, there are tasks which require high recall, e.g., matching as a first step of an interactive merge process, and others which require high precision, e.g., automatic matching for autonomously connecting two web services.

We consider below these two possibilities, starting from the latter.

9.4.1 Aggregating Measures

(Ehrig 2007) provided an analysis of the different needs for evaluation depending of the considered applications. We have applied this technique to the requirements of Table 3.1 (p. 59). As a matter of fact, it can be rewritten with respect to the measurements developed in this chapter. We used this technique to design Table 9.5. This table is slightly more detailed than Table 3.1 because it uses three values instead of two. Here ‘low’ corresponds to not relevant, ‘high’ corresponds to relevant and ‘medium’ corresponds to an intermediate position. Therefore, different *application profiles* could be established to explicitly compare matching algorithms with respect to certain tasks.

Table 9.5 Application requirements of Table 3.1 reinterpreted as measurement weights.

Application	speed	automatic	precision	recall
Ontology evolution (Sect. 1.1)	medium	low	high	high
Schema integration (Sect. 1.2)	low	low	high	high
Catalogue integration (Sect. 1.2)	low	low	high	high
Data integration (Sect. 1.2)	low	low	high	high
P2P information sharing (Sect. 1.4)	high	low	medium	medium
Web service composition (Sect. 1.5)	high	high	high	low
Multi agent communication (Sect. 1.6)	high	high	high	medium
Context matching in ambient computing (Sect. 1.6)	high	high	high	medium
Semantic web browsing (Sect. 1.7)	high	medium	medium	low
Query answering (Sect. 1.7)	high	medium	high	medium

Such a table can be useful for aggregating the measures corresponding to each of these aspects with different weights or to have an ordered way to interpret evaluation results.

Different measures suit different evaluation goals. If we want to improve systems, it is best to have as many indicators as possible. However, in order to single out the best system, it is generally better to focus on the very relevant factors. This can be achieved by only selecting the few very relevant factors, e.g., speed and precision for query answering, or by aggregating measures in relation with their relevance.

For aggregating measures depending on a particular application, it is possible to use weights corresponding to the values of Table 9.5, and thus respecting the importance of each factor. Weighted aggregation measures, such as those presented in Sect. 7.4 (weighted sum, product or average), can be used.

F-measure is already an aggregation of precision and recall. It can be generalised as a harmonic mean, for any number of measures. This requires us to assign every measurement a weight, such that these weights sum to 1. Obviously, the weights have to be chosen carefully depending on the goal.

Definition 9.15 (Weighted harmonic mean) Given a reference alignment R , a set of measures $(M_i)_{i \in I}$ provided with a set of weights $(w_i)_{i \in I}$ between 0 and 1, the weighted harmonic mean of some alignment A is

$$W(A, R) = \frac{\sum_{i \in I} w_i}{\sum_{i \in I} \frac{w_i}{M_i(A, R)}}.$$

Example 9.16 (Weighted aggregation of evaluation measures) Consider that we need to choose among two available systems S_1 and S_2 , delivering A_1 and A_2 , for an application, such as schema integration, peer-to-peer system or query answering. We apply weights corresponding to the criteria of Table 9.5. The weights are *high* = 5, *medium* = 3 and *low* = 1. They are normalised (so as to sum to 1.) for each kind of application. The performance of the systems with regard to the criteria are given in the following table as well as the resulting harmonic means for each pair system \times application:

	Speed	Automatic	Precision	Recall	S_1	S_2
Speed					.8	.5
Automatic					1.	1.
Precision					.7	.9
Recall					.8	.8
Schema integration	.08	.08	.42	.42	.77	.81
Peer-to-peer system	.42	.08	.25	.25	.79	.66
Query answering	.31	.19	.31	.19	.80	.72

Those who need a matching system for a peer-to-peer or query answering application should choose system S_1 (.79 and .80 are better than .66 and .72) and those who want to use it for schema integration should use system S_2 (.81 is better than .77). The importance of speed in the two first applications outweighs the relatively lower precision.

9.4.2 *Evaluation Setting*

Application-specific evaluation can also be carried out by having a specific evaluation setting. This is similar to the task-based evaluation used in ontology learning (Porzel and Malaka 2004). It has the advantage of being more realistic than artificial test beds and of providing very specific information, but the drawback is that it has to be changed for each different application.

An application-specific evaluation has to start with a selection of the task corresponding to the application, as described in Sect. 9.2.1. It is moreover useful to set up experiments which do not stop at the delivery of alignments but carry on with the particular task. This is especially true when there is a clear measure of success of the overall task. Such a setting assists in focussing on the most useful issues for the task. For instance, it may be the case that the gain in accuracy in one algorithm over another is not useful for the task while the gain in speed of the latter really matters. If no clear measure is available, then using a weighted aggregation measure as suggested above would help.

An early attempt to introduce application-specific measures was made in the library test case (a variation of the cultural heritage case in Sect. 1.2) of OAEI 2008 (Isaac et al. 2008). The organisers had to evaluate ontology matching for the purpose of reindexing the catalog of the Dutch national library, indexed by two different thesauri. The organisers only had a partial reference, obtained by pooling the results given by participants and asking librarians to compare these results.

In addition to assessing this theoretical precision and recall, they assessed the quality of the alignments by using them in reindexing. A proportion of the books was annotated by both thesauri, so it was possible to use the alignments to transform the annotations in one of the thesauri and to compare the result with the actual annotations (with precision and recall). Doing so assessed the alignments relative to the real task on which they had to be applied, instead of absolutely: it ignored correspondences which are never used and gave more importance to those which were used often. The results were actually lower than the absolute precision and recall.

However, the organisers did not stop there and asked actual librarians to evaluate the transformed annotations. This allowed for computing new precision and recall values which this time were still not as high as the theoretical precision and recall, but far above the previous ones. Indeed, even if the transformed annotations were not those of the corpus, they were not necessarily incorrect. This scenario and others are presented in (Isaac et al. 2009).

Nevertheless, it is extremely difficult to determine the evaluation value of the matching process independently. The effects of other components of the overall application have to be carefully filtered out.

9.5 Summary

As noted before, it is very difficult to know *a priori* the quality to expect from a matching system. For that purpose, evaluation of matching systems must be performed. The evaluation must apply to all aspects of these systems: their availability, their capacity to provide accurate alignments in a desirable time, etc.

At least two difficulties arise when evaluating matching systems. Matching tasks are so different that a system can perform very well on some data and not that well on some other. It is thus necessary that evaluation data sets be as different as possible and that results are kept separate so that someone with a particular task can choose a system that performs adequately on this task. The second difficulty is the choice of evaluation criteria. As mentioned in Sect. 9.3.2, precision and recall are not always appropriate for ontology matching and must be improved to account for the semantics of ontologies.

Over several years, the Ontology Alignment Evaluation Initiative (Sect. 9.1.3) has become the reference evaluation exercise for ontology matching. It provides quality benchmark suites that system designers can use for training their systems and organises evaluation campaigns for comparing systems. The main purpose of OAEI is to improve the quality of ontology matching algorithms by continuous comparison with other new methods. This seems to have been successful (Euzenat et al. 2011).

Once a matcher has been evaluated and selected, it has to be put to work. This is presented in the following chapters, discussing how alignments can be handled and represented (Chap. 10), how users can be involved in the matching process (Chap. 11), and how resulting alignments can be processed (Chap. 12).

Part IV

Representing, Explaining, and Processing

Alignments

Chapter 10

Frameworks and Formats: Representing Alignments

Once matching has been performed, the resulting alignments are usually used in a wider context than a matching system itself. Several proposals have been made for representing the alignments and exchanging them among tools. This chapter presents some frameworks and formats for doing so. In particular, we address the following aspects:

- Formats and languages that enable the syntactic expression and the manipulation of alignments and that can be used for exchanging alignments across applications (Sect. 10.1).
- Metadata that helps communicate and share correspondences and alignments by qualifying them better for further reuse (Sect. 10.2).
- Frameworks that provide a wider set of operations for manipulating alignments. These frameworks and languages are usually not concerned with the way alignments are found. They, at best, define a match operation which generates alignments. However, they allow plugging in matching methods whose alignments can be manipulated (Sect. 10.3).

We consider these alignment representations independently from their usage, which will be presented in Chap. 12. However, there is a strong relation between them.

10.1 Alignment Formats

Ontology alignments are sets of relationships between ontology entities. Alignments can be expressed in various languages. For instance, the two relations $=$ and \leq may be expressed in OWL (Sect. 10.1.2) through `owl:equivalentClass` and `rdfs:subClassOf`, but they can also be expressed in SKOS (Sect. 10.1.7) through `skos:exactMatch` and `skos:broaderMatch`. Other applications may mandate a different form like views in databases, mediators in web services frameworks or even merged ontologies. The advantage of these latter representations is that they can be processed.

However, application-specific output is not particularly interoperable. It is not easy to transform a database view into OWL axioms or SKOS statements into ODE-Dialect (Corcho and Gómez-Pérez 2007). Indeed, when an alignment is expressed in OWL, it is only possible to ‘merge’ two OWL ontologies; it cannot easily be used to transform queries. Moreover, such formats are not easy to share and retrieve (Sect. 3.8) nor to manipulate (see Sect. 3.7), e.g., for merging the results of several matchers, if they do not use a format that supports such manipulations.

As Chap. 8 indicates, many matching systems deliver alignments. Applications could use this property for replacing a matcher by another one or combining several of them. Unfortunately, the alignment is very often output as a mere list of pairs, HTML table or similarity matrix. We consider here what could be a proper format in order for these systems to interoperate.

Hence, in order to avoid early commitment to a particular type of usage, it is preferred to keep the alignments in a declarative language. Such a language allows for manipulating and composing alignments as well as for generating the required representation (OWL, SKOS and others) when necessary. Using a neutral and declarative representation provides the opportunity to distribute and share alignments among applications. This is why, though in the remainder of this section we consider how to represent alignments in different formats and languages, elsewhere in this book, only ‘alignments’ are considered.

We briefly present various formats that have been proposed so far for expressing relations between ontologies. We mostly compare these formats on the basis of their syntax. An earlier deeper analysis of some of these in terms of semantics and expressiveness is provided in (Serafini et al. 2005; Zimmermann and Euzenat 2006).

For the purpose of a uniform comparison of these formats, we use an example that extends the one of Fig. 2.9. It expresses that

- a Science book in the left-hand side ontology corresponds to an Essay whose subject is an instance of Science in the right-hand side ontology,
- a Pocket book in the left-hand side ontology corresponds to a Volume whose size is less than 14 in the right-hand side ontology,
- a Book which has politics as a topic in the left-hand side ontology corresponds to a Politics essay in the right-hand side ontology, and
- a Writer in the left-hand side ontology is someone who has authored a volume in the right-hand side ontology.

We will consider that the first ontology is identified by the XML &onto1; entity or the <http://book.ontologymatching.org/example/culture-shop.owl> URL and the second ontology is identified by the XML &onto2; entity standing for the <http://book.ontologymatching.org/example/library.owl> URL.

Three of these extra correspondences are displayed in Fig. 10.1. When translated into first-order logic, these can be represented as follows:

$$\forall x, \quad \text{Pocket}(x) \equiv \text{Volume}(x) \wedge \text{size}(x, y) \wedge y \leq 14$$

$$\forall x, \quad \text{Science}(x) \equiv \text{Essay}(x) \wedge (\forall y, \text{subject}(x, y) \Rightarrow \text{Science}(y))$$

$$\forall x, \quad Book(x) \wedge topic(x, politics) \equiv Politics(x)$$

$$\forall x, \quad Writer(x) \Leftarrow \exists y, \quad author(y, x)$$

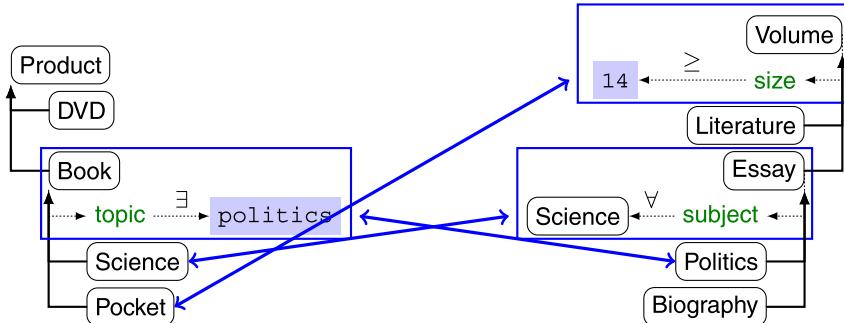


Fig. 10.1 Three correspondences that can be expressed in an elaborate alignment format.

Let us discuss the main formats available for expressing such alignments.

10.1.1 MAFRA Semantic Bridge Ontology (SBO)

MAFRA (Pinto da Silva 2004; Mädche et al. 2002) stands for MAppling FRamework¹ (see also Sect. 10.3.4). It is a system for extracting mappings from ontologies and executing them as data transformation from one ontology to another one. The system was first designed to work with the DAML + OIL language, an ancestor of OWL.

MAFRA does not define a real exchange format for ontology alignment. Rather, it provides an ontology, called the Semantic Bridge Ontology. The instantiation of this ontology is an ontology mapping document. The serialisation of this format has not been described in detail in documents so we freely use our own transcription.

The main concepts in this ontology are SemanticBridges and Services. A SemanticBridge is tied to the Services that are able to implement the bridge as a data transformation. A Service can be thought of as a function: $f : Arg^n \rightarrow Arg^m$ that maps tuples of arguments into tuples of arguments. It can be identified by a URI, for instance. The arguments are typed and can be ontology concepts, property paths, literals or arrays of these.

SemanticBridges, which in turn can be ConceptBridges or PropertyBridges, express a relation between two sets of entities by composing elementary services that are applied to them. For instance, a SemanticBridge between two ontologies can map those Volumes with size larger than 14 to Pocket books in the following way:

¹This is also the name of a city in Portugal featuring a rich palace.

```

ConceptBridge: Volume2Pocket
  x: <o2#Volume>; o2:size >= 14 -> <o1#Pocket>
ConceptBridge: Book2Politics
  x: <o1#Book>; o1:topic == 'politics' -> <o2#Politics>

```

Entities to be mapped are identified within the ontology (instances) through a path. Paths serve the dual purposes (i) of navigating within the ontology structure and (ii) of providing the context for further characterising the concerned entities. In this context, paths play exactly the same role as in Xpath (Clark and DeRose 1999). They are further enriched with conditions. In the example above, `<o2#Volume>; o2:size >= 14` is a path with condition that the final step size has a value inferior to 14.

An ontology mapping document satisfying the semantic bridge ontology is a collection of such bridges plus information on the concerned ontologies, as well as constraints: for example, the *exclusivity* conditions can be used for expressing that an entity cannot be mapped by more than one bridge rule.

The semantic bridge ontology provides a framework and a format for expressing alignments. This format is used as output from ontology matchers and input for data transformations.

The format provided by the Semantic Bridge Ontology is not very clear since the language is described in UML. This is a minor problem that could be solved by exposing some RDF/XML format (a previous version of the framework had been described as a DAML ontology (Mädche et al. 2002)). Moreover, this format is a relatively complex language that is tied to the MAFRA architecture (Sect. 10.3.4). It does not separate the declarative aspect of relations from the more operational aspects of services: relations are described with regard to services able to implement them. The services can be arbitrary small, such as string concatenation, or large, such as implementing a full alignment by a program. On the one hand, this guarantees that these alignments can be used: SBO-documents can readily be used as data translation. On the other hand, this does not favour the use of these alignments in other ways, for instance, for merging ontologies or mediating queries.

10.1.2 OWL

OWL (Sect. 2.2) can be considered as a language for expressing correspondences between ontologies. As a matter of fact, the `equivalentClass` and `equivalentProperty` primitives have been introduced for relating elements in ontologies describing the same domain. This use has been documented by the W3C best practices working group (Uschold 2005). Moreover, these primitives are only shorthands for other primitives, e.g., `subClassOf`, `subPropertyOf`, which already relate entities. For example, the following OWL ontology fragment

```

<owl:Property rdf:about="#onto1;#author">
  <owl:equivalentProperty rdf:resource="#onto2;#author" />
</owl:Property>

<owl:Class rdf:about="#onto1;#Book">

```

```

<owl:equivalentClass rdf:resource="#onto2;#Volume" />
</owl:Class>

<owl:Class rdf:about="#onto2;#title">
  <owl:subClass = "#onto1;#name" />
</owl:Class>

```

can be seen as an alignment expressing the equivalence of properties author and author, the equivalence of classes Book and Volume and the coverage of property title in the second ontology by name in the first one. Moreover, any ontology, as soon as it involves entities from different ontologies, expresses alignments. For instance, the following OWL ontology fragment

```

<owl:Class rdf:ID="#onto1;#Science">
  <owl:equivalentClass>
    <owl:Class>
      <owl:subClassOf rdf:resource="#onto2;#Essay" />
      <owl:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#onto2;#subject" />
          <owl:allValuesFrom rdf:resource="#onto2;#Science" />
        </owl:Restriction>
      </owl:subClassOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="#onto2;#Writer">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#onto1;#hasWritten" />
      <owl:minCardinality
        rdf:datatype = "&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>

```

expresses that a Science book in the first ontology is equivalent to an Essay whose subject is an instance of the Science class in the second one and that those who have written at least one thing in the first ontology are Writer in the second one.

Not surprisingly, the OWL language can be used as an alignment expression language. However, using it this way has some drawbacks:

1. It forces the use of a particular ontology language: OWL. It is still possible to relate in this way ontologies that are expressed in other languages without benefiting from the construction of complex terms. However, the alignment will not benefit from the content of the ontologies themselves.
2. It mixes correspondences and definitions. This is a problem for the clarity of alignments as well as for lightweight applications which do not want to interpret the OWL language.
3. It is interpreted only in the framework of a global interpretation of one OWL theory (see Sect. 2.5.3). It is difficult to use this expression for only importing data expressed under one ontology into another one because this application requires sorting out definitions from correspondences.
4. It cannot express data transformation.

Other languages have been designed for overcoming these problems. For example, SKOS solves the first problem, but introduces its own language, SWRL solves

the second problem and C-OWL attempts to solve the third problem. These languages are presented hereafter.

10.1.3 Contextualized OWL (C-OWL)

C-OWL is an extension of OWL to express mappings between heterogeneous ontologies (Bouquet et al. 2004b). The new constructs in C-OWL, with respect to OWL, are called *bridge rules*, and they allow the expression of relations between classes, relations and individuals interpreted in heterogeneous domains.

Bridge rules are oriented correspondences, from a source ontology o to a target ontology o' . They use five relations: more general (\geq), more specific (\leq), equivalent ($=$), disjoint (\perp) and overlap (\circlearrowright). These relations are always applied to named entities. Bridge rules are always interpreted from the standpoint of the target ontology. They express how the target ontology translates the source ontology in itself.

Bridge rules are expressed separately from the ontologies they refer to. The examples considered here are given below in C-OWL XML syntax:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY cowl "http://www.itc.it/cowl#" >
  <!ENTITY onto1 "http://book.ontologymatching.org/example/culture-shop.owl" >
  <!ENTITY onto2 "http://book.ontologymatching.org/example/library.owl" >
]>
<rdf:RDF
  xmlns      = "&cowl;""
  xmlns:cowl = "&cowl;""
  xmlns:owl   = "&owl;""
  xmlns:rdf   = "&rdf;""
>
<cowl:Mapping>
  <cowl:sourceOntology>
    <cowl:Ontology rdf:about="&onto1;" />
  </cowl:sourceOntology>
  <cowl:targetOntology>
    <cowl:Ontology rdf:about="&onto2;" />
  </cowl:targetOntology>
  <cowl:bridgeRule>
    <cowl:Intro>
      <cowl:source>
        <owl:Class rdf:about="&onto1;#Book"/>
      </cowl:source>
      <cowl:target>
        <owl:Class rdf:about="&onto2;#Volume"/>
      </cowl:target>
    </cowl:Intro>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Onto>
      <cowl:source>
        <owl:Class rdf:about="&onto1;#name"/>
      </cowl:source>
      <cowl:target>
        <owl:Class rdf:about="&onto2;#title"/>
      </cowl:target>
    </cowl:Onto>
  </cowl:bridgeRule>
</cowl:Mapping>
```

```

</cowl:bridgeRule>
<cowl:bridgeRule>
  <cowl:Equivalent>
    <cowl:source>
      <owl:Class rdf:about="#ontol;#author"/>
    </cowl:source>
    <cowl:target>
      <owl:Class rdf:about="#onto2;#author"/>
    </cowl:target>
  </cowl:Equivalent>
</cowl:bridgeRule>
<cowl:bridgeRule>
</cowl:Mapping>
</rdf:RDF>

```

The C-OWL proposal can express relatively simple alignments: no constructed classes are expressed, only named classes are used. The more expressive part resides in the relations used by the mapping. These alignments have a clear semantics, however it is given from a particular standpoint: that of the target ontology. C-OWL is based on the OWL language but relatively independent from this language which is confined at expressing entities (the alignment part being specific).

10.1.4 SWRL and RIF

It is sometimes not enough to be able to express entity definitions; sometimes expressing rules is a more convenient expression mean. Moreover, rules can bring more expressiveness. SWRL (Semantic Web Rule Language) (Horrocks et al. 2004) is a rule language for the semantic web. It extends OWL with an explicit notion of rule that is interpreted as first order Horn clauses. These rules can be understood as correspondences between ontologies, especially when elements from the head and the body are from different ontologies. These correspondences are oriented as in C-OWL.

SWRL mixes the vocabulary from RuleML for exchanging rules with the OWL vocabulary for expressing knowledge. It defines a rule (`ruleml:imp`) with a body (`ruleml:body`) and head (`ruleml:head`) parts.

```

<ruleml:imp>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="#ontol;#Book" />
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="#ontol;#topic">
      <ruleml:var>p</ruleml:var>
      <owlx:DataValue owlx:datatype="#xsd:string">politics</owlx:DataValue>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom swrlx:property="#onto2;#Politics">
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>

```

```

<ruleml:imp>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="#onto2;#Volume" />
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom swrlx:property="#onto2:#size">
      <ruleml:var>p</ruleml:var>
      <ruleml:var>q</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:builtinAtom swrlx:builtin="#greaterThanOrEqual">
      <owlx:DataValue owlx:datatype="#xsd:#int">14</owlx:DataValue>
      <ruleml:var>q</ruleml:var>
    </swrlx:builtinAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom swrlx:property="#onto1:#Pocket">
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>

```

The first rule expresses that a Book in the first ontology with politics as a value of its topic attribute is a Politics book in the second ontology. The second rule expresses that Volumes in the second ontology whose size is less than 14 are Pocket books in the first ontology.

The introduction of variables within constructs of the OWL language provides more expressiveness to the language. In particular, it allows the expression of what was called role-value maps in description logics or feature path equations in feature algebras (Smolka 1992). Of course, all the constructions available in OWL are usable in SWRL as well. SWRL also provides a set of built-in predicates on various data types provided by XML Schema as well as operators on collections, such as count.

SWRL rules can be used for expressing the correspondences between ontologies. These correspondences are expressed between formulas and interpreted as Horn clauses. They have the advantage over genuine OWL of being well identified as rules and are easier to manipulate as an alignment format than OWL, which is also used to express ontologies.

As in the OWL case, these rules have the drawback of forcing the use of OWL and are interpreted as merging ontologies. Again, the expression of a rule, such as the one above, freezes the use that can be made: the rule will help in considering some Books of the first ontology as Politics books in the second ontology. However, the rules work as a set of logical rules, not rewrite rules, so they can be used for merging, but not transforming ontologies.

The two examples above can be expressed in the presentation syntax of the Rule Interchange Format (RIF) (Boley and Kifer 2010):

```

Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(pred <http://www.w3.org/2007/rif-built-in-predicate#> )
  Prefix(onto1 <http://book.ontologymatching.org/example/culture-shop.owl#>)
  Prefix(onto2 <http://book.ontologymatching.org/example/library.owl#>)

Group (
  Forall ?x (

```

```

rdf:type( ?x onto2:Politics ) :-
    And( rdf:type( ?x onto1:Book ) onto1:topic( ?x "politics" ) )

Forall ?x (
    rdf:type( ?x onto1:Pocket ) :-
        And( rdf:type( ?x onto2:Volume )
            onto2:size( ?x ?s )
            External( pred:numeric-lower-than( ?s, 14 ) )
    )
)

```

10.1.5 Alignment Format

The Alignment format (Euzenat 2004) is simpler than most of the alignment representations presented here. It aims at being producible by most matching tools, yet preserving the capability to handle complex alignment definitions.

The alignment description is an envelope in which the correspondences are grouped. It expresses metadata about alignments (Sect. 10.2). Alignments are made of:

References to matched ontologies;

A set of correspondences which expresses the relation holding between entities of the first ontology and entities of the second ontology;

Level used for characterising the type of correspondence (see next);

Arity (default 1:1) (denoted by 1 for injective and total, ? for injective, + for total and * for none, with each sign concerning one mapping and its converse): ?:?, ?:1, 1?:1, 1:1, ?:+, +?:1, 1:+, +:1, +:+, ?:*, ?:?, 1:*, *:1, +:*, *:+, *;* (see also Sect. 2.5.2). These assertions could be provided as input (or constraint) for a matching algorithm or as a result by the same algorithm.

The Alignment format can be extended by new metadata tags² which may apply to both alignments or correspondences. Metadata are represented as key-value pairs whose keys are URIs for faster retrieval and whose values are limited to strings. They are preserved through the serialisation of alignments in RDF.

Support for correspondences follows Definition 2.10. They are expressed by:

entity1: the first matched entity;

entity2: the second matched entity;

relation: the relation holding between the two entities. It is not restricted to the equivalence relation, but can be more sophisticated, e.g., subsumption, incomparability, or even some fuzzy relation. The default relation is equivalence.

strength: the confidence that the correspondence under consideration holds. The measure should belong to an ordered set \mathcal{E} including a maximum element \top and a minimum element \perp ; for instance, a float value between 0 and 1. The default strength is \top .

²Metadata identifiers are registered at <http://alignapi.gforge.inria.fr/labels.html>.

id: an identifier for the correspondence.

A full example of the level 0 Alignment format in RDF is as follows:

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY xsd      "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY onto1    "http://book.ontologymatching.org/example/culture-shop.owl" >
  <!ENTITY onto2    "http://book.ontologymatching.org/example/library.owl" >
]>

<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
           xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>

<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <time>7</time>
  <method>fr.inrialpes.exmo.align.impl.method.StringDistAlignment</method>
  <onto1>
    <Ontology rdf:about=&quot;&#x26;onto1;&quot;>
      <location>file:examples/rdf/onto1.owl</location>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about=&quot;&#x26;onto2;&quot;>
      <location>file:examples/rdf/onto2.owl</location>
    </Ontology>
  </onto2>
  <map>
    <Cell>
      <entity1 rdf:resource='&#x26;onto1;#Book'/>
      <entity2 rdf:resource='&#x26;onto2;#Volume'/>
      <relation>&lt;</relation>
      <measure rdf:datatype='&#x26;xsd;float'>0.63636363636364</measure>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='&#x26;onto1;#name'/>
      <entity2 rdf:resource='&#x26;onto2;#title'/>
      <relation>></relation>
      <measure rdf:datatype='&#x26;xsd;float'>1.0</measure>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

It describes a many-to-many level 0 alignment between two bibliographic ontologies. It contains two correspondences in which Book in the first ontology is less general than Volume in the second one and name in the first ontology is more general than title in the second one. These correspondences use the less and more general relations and a confidence measure .64 in the former case and 1. in the latter.

The Alignment format has been designed for offering a common format to different needs. Depending on the expressiveness of the matched entities, it offers several alignment levels which correspond to different options for expressing entities:

Level 0: These alignments relate entities identified by URIs. Any algorithm can deal with such alignments. This first level of alignment has the advantage to not depend on a particular language for expressing these entities. On this level, the

matched entities may be classes, properties or individuals. However, they also can be any kind of a complex term that is used by the target language as soon as it is identified by a URI.

Level 1: These alignments replace pairs of entities by pairs of sets (or lists) of entities. A level 1 correspondence is thus a slight refinement of level 0. It can be easily parsed and is still language independent.

Level 2 (L): More general correspondence expressions may be useful. For instance, (Masolo et al. 2003) provides bridges from an ontology of services to the currently existing semantic web service description languages in first order logic. These kinds of correspondences can be expressed with level 2 alignments. These are no longer language independent and require the knowledge of the language used for parsing the format. In this case correspondences can be expressed between formulas, queries, etc. EDOAL (Sect. 10.1.6) is an expressive level 2 language.

The Alignment format has been given an OWL ontology and a DTD for validating it in RDF/XML. It can be manipulated through the Alignment API which is presented next in Sect. 10.3.6. It has been used as the format for the OAEI evaluation campaigns (Sect. 9.1.3) so many different tools are able to output it, e.g., oMap (Sect. 8.3.4), FOAM (Sect. 10.3.7), OLA (Sect. 8.3.8), Falcon-AO (Sect. 8.3.9), and HCONE (Sect. 8.1.19).

Finally, the Alignment format allows the expression of alignments without commitment to a particular language. It is not targeted towards a particular use of the alignments and offers generators for many other formats. However, in contrast to the languages presented so far, this format does not offer much expressiveness. One of its good features is its openness which allows the introduction of new relations and, if necessary, new types of expressions while keeping the compatibility with poorly expressive languages.

10.1.6 Expressive and Declarative Ontology Alignment Language (EDOAL)

EDOAL (Expressive and Declarative Ontology Alignment Language) is a level 2 language for the Alignment API. It has been formerly known as the SEKT mapping language (de Brujin et al. 2004) or OMWG (Euzenat et al. 2007b). It is an expressive language independent from any ontology language, even if it largely borrows constraints provided in description logics or the OWL language. This independence allows for the representation of alignments between heterogeneous and weak representations, such as a thesaurus and a relational database. Thus, EDOAL has a middle man position: it is independent from any particular language but expressive enough for covering a large part of the other languages.

EDOAL extends the Alignment format in order to capture more precisely correspondences between heterogeneous ontological entities. To achieve this, EDOAL uses the following features (see Table 10.1):

Construction of entities from other entities can be expressed through algebraic operators. Constructing entities is one way to overcome the shallowness of some ontologies.

Restrictions can be expressed on entities in order to narrow their scope. Narrowing the scope of an entity makes it possible to more precisely align this entity with the corresponding one in the other ontology.

Transformations of property values can be specified. Property values using different encodings or units can be aligned using transformations. The current version of EDOAL only supports a limited kind of transformations.

Table 10.1 EDOAL expression constructors.

Entity	Constructors	Restrictions
Class	and, or, not	domain, type, value, cardinality
Property	and, or, not, compose	domain, type, value
Relation	and, or, not, compose inverse, symmetric, reflexive, transitive	domain, codomain
Instance		

The interpretation of the language follows the rules of a classical ontology language (Sect. 2.2.2).

For instance, that an Autobiography whose size is less than 14 is the same thing as a Pocket book whose topic is the creator can be expressed by:

```

<Cell rdf:about="pocketbook">
<entity1>
<edoal:Class>
  <edoal:and rdf:type="Collection">
    <edoal:Class rdf:about="#o;Pocket"/>
    <edoal:AttributeValueRestriction>
      <edoal:onAttribute>
        <edoal:Property rdf:about="#o;creator"/>
      </edoal:onAttribute>
      <edoal:comparator rdf:resource="#edoal>equals"/>
      <edoal:value>
        <edoal:Property rdf:about="#o;topic"/>
      </edoal:value>
    </edoal:AttributeValueRestriction>
  </edoal:and>
</edoal:Class>
</entity1>
<entity2>
<edoal:Class>
  <edoal:and rdf:type="Collection">
    <edoal:Class rdf:about="#o';Autobiography"/>
    <edoal:AttributeValueRestriction>
      <edoal:onAttribute>
        <edoal:Property rdf:about="#o';size"/>
      </edoal:onAttribute>
      <edoal:comparator rdf:resource="#edoal=less-than"/>
      <edoal:value>
        <edoal:Literal edoal:type="#xsd;integer" edoal:string="14" />
      </edoal:value>
```

```

    </edoal:AttributeValueRestriction>
    </edoal:and>
</edoal:Class>
</entity2>
<measure rdf:datatype='&xsd;float'>1.</measure>
<relation>=</relation>
</Cell>

```

It can also be expressed that in the context of a Book, the creator is the author:

```

<Cell rdf:about="pocketbook">
<entity1>
<edoal:Relation>
    <edoal:and rdf:parseType="Collection">
        <edoal:Class rdf:about="&o';creator"/>
        <edoal:DomainRestriction>
            <edoal:class rdf:about="&o;Book"/>
        </edoal:DomainRestriction>
    </edoal:and>
</edoal:Relation>
</entity1>
<entity2><edoal:Relation rdf:about="&o';author"/></entity2>
<measure rdf:datatype='&xsd;float'>1.</measure>
<relation>=</relation>
</Cell>

```

Moreover, EDOAL is also able to express transformations between entities, e.g., unit transformations or any type of defined function. For that purpose, correspondences may contain transformation elements, and EDOAL allows for defining variables used to denote parts of the matched entities in transformations. For instance, to the previous example could be added a transformation meaning that the foaf:name of the creator is the concatenation of the vcard:firstname, vcard:middleinitial and vcard:lastname of the author:

```

<edoal:transformation>
    <edoal:Transformation edoal:direction="o->">
        <edoal:entity1>
            <edoal:Property rdf:about="&foaf;name">
        </edoal:entity1>
        <edoal:entity2>
            <edoal:Apply edoal:operator="&edoal;concat">
                <edoal:arguments rdf:parseType="Collection">
                    <edoal:Property rdf:about="vcard:firstname" />
                    <edoal:Literal edoal:type="&xsd:string" edoal:string=" " />
                    <edoal:Property rdf:about="vcard:middleinitial" />
                    <edoal:Literal edoal:string=". " />
                    <edoal:Property rdf:about="vcard:lastname" />
                </edoal:arguments>
            </edoal:Apply>
        </edoal:entity2>
    </edoal:Transformation>
</edoal:transformation>

```

This type of information is very useful at the instance level, i.e., when one wants to interlink (Sect. 12.4) or integrate data. (Sect. 12.3). In addition, EDOAL allows for representing correspondence patterns (see Sect. 6.1.4). It is supported by the Alignment API through providing an API for manipulating EDOAL alignments, parsers, and renderers in RDF, OWL, and SPARQL.

10.1.7 SKOS

SKOS (Miles and Bechhofer 2009) stands for Simple Knowledge Organisation System. The SKOS core vocabulary is an RDF Schema aiming at expressing relationships between lightweight ontologies, e.g., folksonomies or thesauri.

The SKOS mapping vocabulary offers further properties for linking concepts of different SKOS thesauri.

Concept and Relation Descriptions

SKOS allows the identification of concepts in lightweight ontologies through the `skos:Concept` class. It provides various ways of presenting the concept with labels in several languages and alternate labels and symbols. It also provides the opportunity to add various notes and informal definitions to the concept. In the example below, Book is a concept with some information about it, such as the way it is named in different languages and its definition (similar to glosses in WordNet).

```
<skos:Concept rdf:about="#Book">
  <skos:prefLabel>Book</skos:prefLabel>
  <skos:altLabel xml:lang="fr">Livre</skos:altLabel>
  <skos:definition>A book is a set of sheets of papers bound together so that the content printed on them can be consulted in sequence.</skos:definition>
  <skos:editorialNote>This is not an official definition</skos:editorialNote>
  <skos:broader rdf:resource="#ontol:#Product"/>
  <skos:narrower rdf:resource="#ontol:#Pocket"/>
<skos:Concept/>
```

SKOS also provides relationships that express relations between SKOS concepts. For instance, a term used in a thesaurus may be broader than another. Relations between concepts assert the relative inclusion of concepts as broader or narrower terms as well as other informal relations. The above example displays the Book concept that is narrower than Volume but broader than Critics. It is also related to Work.

Mapping Vocabulary

In addition, SKOS defines a mapping vocabulary which aims at expressing relationships across thesauri. It is thus closer to an alignment vocabulary. It is based on the properties presented in Table 10.2.

Table 10.2 SKOS mapping properties.

property	subPropertyOf	domain	range	inverse	property
broaderMatch	broader	concept	concept	narrowerMatch	
relatedMatch	related	concept	concept	relatedMatch	symmetric
closeMatch		concept	concept	closeMatch	symmetric
exactMatch	closeMatch	concept	concept	exactMatch	symmetric, transitive

The following example expresses that the Book concept in ontology onto1 is narrower than Volume but broader than Critics in onto2. It is also related to Work.

```
<skos:Concept rdf:about="#onto1;#Book">
  <skos:broaderMatch rdf:resource="#onto2;#Volume"/>
  <skos:narrowerMatch rdf:resource="#onto2;#Critics"/>
  <skos:relatedMatch rdf:resource="#onto2;#Work"/>
<skos:Concept/>
```

SKOS has the advantage of being a lightweight vocabulary defining from the ground a rich collection of relations between entities. Since it uses URIs for referring to objects it is fully integrated in the semantic web architecture and is not committed to a particular language. In fact, one of the main advantage of SKOS is that it lifts any kind of organised description into an easily usable set of classes. The relation part has the advantage of being very general but the drawback is that it lacks formal semantics. However, more semantics on these terms may be introduced by using the OWL vocabulary.

Like other formats which do not separate the ontologies from the correspondences, SKOS, in its most convenient form, mixes the highest power of RDF Schema and the expression of the alignments. This type of extensibility, through RDF Schema, prevents any non RDF Schema understanding application from fully grasping the SKOS content.

10.1.8 Comparison of Existing Formats

The formats we have considered so far may be globally compared (see Table 10.3). For that purpose, a set of criteria is applied to these different formats:

Web compatibility is the capacity of the format to be manipulated on the web.

This involves its possible expression in XML, RDF and/or RDF/XML, as well as the possibility to identify entities by URIs. This should, in principle, enable the extensibility of the format by introducing new properties as well as the referencing of particular correspondences individually. This aspect is covered by the RDF/XML and URI criteria of Table 10.3.

Language independence is the ability to express alignments between entities described in different languages. This is often related to the use of URIs. In fact, language independence is mostly related to simplicity. This aspect is covered by the Language and Model criteria of Table 10.3.

Simplicity is the capacity to be dealt with in a simple form by simple tools. In particular, for example, requiring inference for correctly manipulating the alignment or requiring that the format covers an important part of some ontology representation language is not a sign of simplicity. A well structured format should help achieve this goal. This aspect is inverse to expressiveness.

Expressiveness is the capacity of the format to express complex alignments. This means that alignments are not restricted to matching entities identified by URIs but can create new entities. The constructions for expressing alignments can be

arbitrary complex. In fact, it can be more complex than the knowledge expressed in the ontologies. This aspect is covered by the Relations, Terms, Type restrictions, Cardinality, Variables and Built-in criteria of Table 10.3.

Extendibility is the capacity to extend the format with specific purpose information in such a way that the tools that use this format are not disturbed by the extensions. Most of the systems presented here exhibit one kind of extendibility tied to the use of RDF which allows any new relation and object to be added. This aspect is covered by the ‘+’ signs in Table 10.3.

Purpose independence is a consequence of various factors expressed. We mention the intended use of each of the formats. It is clear, for instance, that a format designed for data integration, with very precise selection constraints, will rather be difficult to re-use in transforming ontologies. This aspect is covered by the Target application criterion of Table 10.3.

Executability is the capacity to be directly usable in mediators. This means that there are tools available for directly interpreting the format as a program processing knowledge. Executability is rather opposed to language independence. This aspect is covered by the Execution criterion of Table 10.3.

Table 10.3 provides the values of all the formats presented so far for each criterion.

Table 10.3 Summary of characteristics of the presented formats. + means that the system can be extended; Transf stands for transformation. The possible relations for the formats are subclass (sc), subproperty (sp), implication between formulas (imp). The terms concerned by the alignments can be classes (C), properties (P) or individuals (I).

Format	OWL	SBO	C-OWL	SWRL	Alignment	EDOAL	SKOS
Target app.	Merging	Data transf.	Data int.	Data int.	Generic	Generic	Merging
Language	OWL	UML	OWL	OWL	+		RDFS
Model	OWL		OWL+	OWL			
Execution	Logical	Transf	Logical	Logical		Logical	Alg.
RDF/XML	✓		✓	✓	✓	✓	✓
URI	✓		✓	✓	✓	✓	✓
Measures					✓	✓	
Relations	sc/sp		sc/sp	imp	sc/sp+	sc/sp/...	sc/sp
Terms	C/P/I	C/P/I	C/P/I	C/P/I	URI	C/P/I	C/P
Type rest	✓	✓	✓	✓		✓	
Cardinality	✓		✓	✓			
Variables				✓			
Built-in		✓+		✓	+	+	

The difference between these formats lies in the continuum between: (i) very general languages that are easy to understand but which are unable to express complex alignments, e.g., SKOS, level 0 Alignment format, and (ii) very expressive

languages whose semantics dictates their use and which require deep understanding of the language, e.g., OWL, SWRL, C-OWL, MAFRA. EDOAL stands in the middle of this continuum.

While most of the matching algorithms are only able to express the first kind of alignments, both kinds of languages are useful. Most of the expressive formats have a surface heterogeneity due to the languages on which they are based, e.g., UML, OWL, WSML. However, they have very similar features for referring to ontology constructs, such as classes, properties, using logical formula constructors, such as conjunctions, implications, quantifiers, or using data types and collections of built-in operators. Finally, it is surprising that there is not more heterogeneity in these expressive languages given that complexity is a factor of the language used for expressing the ontologies, the language used for expressing the related entities, the semantics given to alignments and the language used for expressing relations.

As a summary, there is no universal format for expressing alignments. The choice of a format depends on the characteristics of the application. We think that there are two factors which can influence this choice: (i) the expressiveness required for the alignments, and (ii) the need to exchange with other applications, especially if they involve different ontology languages.

Formats express the content of alignments. In order to be easily retrieved, alignments may be described through metadata. Principles of these metadata are now presented.

10.2 Alignment Metadata

When an alignment has to be communicated to other parties, it can be annotated so that it will be easier to understand it. Annotations of alignments, or alignment metadata, record useful information for retrieving alignments (see Sect. 3.3) or for explaining them (see Sect. 11.3). Annotations can be carried by alignments or by correspondences individually.

Example 10.1 (Alignment metadata) Here is a sample of metadata associated with an alignment (using the Alignment format tags and externally declared tags such as Dublin core):

dc:date	2009/10/23
align:method	fr.inrialpes.exmo.align.impl.methods.StringDistAlignment
align:time	421
omwg:purpose	Query mediation
dc:creator	John Doe

and correspondence annotations can be:

align:measure	.7768
align:note	'manually validated'

Although any type of annotation may be defined, there are at least three important categories of metadata.

10.2.1 Identification Metadata

Identification metadata describes characteristics of the alignment, such as

- the aligned ontologies,
- the language in which the ontologies are expressed,
- the kind of alignment it is (1:1 or *:*, for instance),
- the initial alignment from which an alignment is derived, if any,
- the application context,
- the properties satisfied by the correspondences (and their proof, if necessary),
- the certificate from an issuing source.

Identification metadata is important for retrieving alignments.

10.2.2 Provenance Metadata

Provenance metadata records how the alignment has been obtained, for instance:

- the algorithm or operation that provided it (or whether it has been provided manually), in case of matchers, it is useful to know the exact version,
- the parameters passed to the generating algorithms,
- the external knowledge sources that may have been used, e.g., lexicons or ontologies,
- the dependency across correspondences,
- the date when the correspondences were created,
- the user who performed the matching operation,
- external references.

This information is already very precious and helps applications in selecting the most appropriate alignments or in explaining alignments. It is thus necessary that ontology matchers be able to generate these metadata. Some useful information like the algorithm used for computing it, the time it took, or the source alignments and the date of matching can generally be recorded automatically.

Figure 10.2 shows the ontology metadata vocabulary (Hartmann et al. 2005) extension to alignments. It includes classes for describing the matching strategy as well as properties satisfied by alignments and their proofs.

Among provenance metadata, justifications are very useful for explaining alignments (see Sect. 11.3). Each correspondence can be assigned one or several justifications that support or infirm the correspondence. We call them *justified correspondences*. For instance, the justified correspondence:

$$\langle e, e', \leq \rangle : \text{justification: } I(e) \subseteq I(e')$$

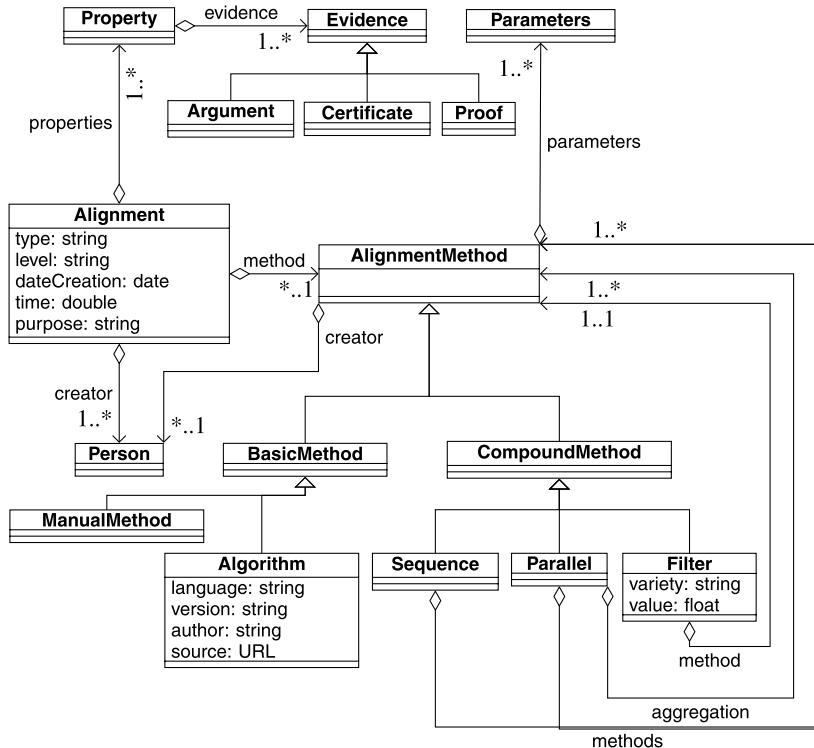


Fig. 10.2 UML characterisation of alignment metadata.

expresses that the correspondence $\langle e, e', \leq \rangle$ is thought to hold because ' $I(e) \subseteq I(e')$ ' is verified. Similarly:

$$\langle e, e', = \rangle : \text{justification: DPLL entailed}$$

expresses that the correspondence $\langle e, e', = \rangle$ is thought to hold because it has been proved by the Davis–Putnam–Longemann–Loveland (DPLL) procedure (Davis and Putnam 1960; Davis et al. 1962).

Justifications may be more complex. For instance, the second justification may involve a full proof of the correspondence and the axioms involved in that proof. This justification information can be found directly within the correspondences or provided on-demand by the matchers to the system requiring explanation (see Sect. 7.4.3).

Matching systems typically combine multiple matchers (see Chap. 7). The final alignment is usually a result of synthesis, abstraction, deduction, and some other manipulations of their results. Thus, users may want to see a trace of the performed manipulations. We refer to them as process traces. Some examples of this kind of information include

- a trace of rules or strategies applied,
- support for alternative paths leading to a single conclusion,
- support for accessing the implicit information that can be made explicit from any particular reasoning path.

Such traces may be expressed in the vocabulary provided in Fig. 10.2. Users may also want to understand why a particular correspondence was not discovered, or why a discovered correspondence was ranked in a particular place, thereby being included in or excluded from the final alignment.

Finally, alignments may rely on external sources of background knowledge. These sources must be clearly identified not simply by naming them but also by identifying which version of a source is used.

10.2.3 Qualification Metadata

Qualification metadata asserts judgements on the alignments or correspondences, such as

- general comments,
- discussions and user comments,
- the confidence in each correspondence,
- the limitations of the use of the alignment,
- the arguments in favour of or against a correspondence (Laera et al. 2007).

Qualification is useful when users publicly share alignments (Noy et al. 2008) as it allows them to annotate alignments with opinions about them and optionally use social networking techniques for collectively improving alignments (see Sect. 11.2). It can also be used by agents for arguing about alignments.

We now consider frameworks taking advantage of some explicit representation of alignments to offer alignment creation and manipulation. Usually such frameworks are open to the addition of new matchers.

10.3 Alignment Frameworks

The matching operation is typically only one of the steps towards the ultimate goal of ontology integration or web services composition, for instance. There exist infrastructures which use alignments as one of their components.

Platforms for integrating matchers and alignment manipulation operations are relatively new. However, they constitute a promising perspective to knowledge engineers and application developers. We can distinguish two types of software in alignment management: (i) the infrastructure middleware, and (ii) the support environments that provide application-specific access to alignments. The two levels may be kept clearly separated (Euzenat 2005) or mixed in a single system (Noy et al.

2008; Mork et al. 2008). We discuss here the latter, the former being addressed in Sect. 12.7. The goal of support environments is to enable users to perform high-level tasks which involve generating, manipulating, composing and applying alignments within the same environment. As in Chap. 8, in this section we did not enforce the terminology of Sect. 2.4 but kept it as used by system designers.

10.3.1 Model Management

Model management (Bernstein et al. 2000; Madhavan et al. 2002; Melnik 2004) aims at providing a metadata manipulation infrastructure in order to reduce the amount of programming required to build metadata driven applications. It has been promoted in databases for dealing with data integration in a generic way. It offers a high-level view of the operations applied to databases and their relations. Model management deals with models which can be related by mappings. A *model* is an information structure, such as XML schema, relational database schema, UML model. Similarly, *mappings* are oriented alignments from one model into another. Technically, a key idea of generic model management is to solve metadata intensive tasks at a high level of abstraction using a concise script. It provides an algebra to manipulate models and mappings. It is generic in the sense that a single implementation should be applicable to the majority of data models and scenarios, e.g., data translation, data integration. However, it is primarily targeting databases. In (Melnik et al. 2005), the following operators are defined:

- $\text{Match}(m, m')$ which returns the mapping a between models m and m' ;
- $\text{Compose}(a, a')$ which composes mappings a and a' into a new one a'' , given that the range of a' is the domain of a ;
- $\text{Confluence}(a, a')$ which merges alignments by union of nonconflicting correspondences, provided as a and a' that have the same domain and range;
- $\text{Merge}(a, m, m')$ which merges two models m and m' according to mapping a ;
- $\text{Extract}(a, m)$ which extracts the portion of model m which is involved in the mapping a ;
- $\text{Diff}(a, m)$ which extracts the portion of model m which is not involved in the mapping a .

A mapping in this context is a function from m to m' . (Melnik et al. 2005) also provides axioms governing these operations. For instance, the merge operation between two models m' and m'' through a mapping a , returns a new model $m = \text{Domain}(a') \cup \text{Domain}(a'')$ and a pair of surjective mappings a' and a'' from m to m' and m'' respectively, such that $a = \text{Compose}(\text{Invert}(a'), a'')$.

A typical example of model management script is as follows:

```

A1 := Match( O1, O2 );
A2 := Match( O2, O3 );
O4 := Diff( O1, A1 );
A3 := Compose( A1, A2 );
O5 := Merge( Extract( O1, A1 ), O3, A3 );
O6 := Merge( O4, O5, Ø );

```

The above example operates with three ontologies. It merges the first one and the last one on the basis of the composition of their alignment with the intermediate one. Finally, it adds the part of the first one that was not brought in the first merge.

There are some model management systems available. In particular, Rondo is a programming platform implementing generic model management (Melnik et al. 2003a, 2003b). It is a standalone program with no editing functions. It is based on conceptual structures which constitute the main Rondo abstractions:

Models such as relational schemas, XML schemas, are internally represented as directed labelled graphs, in which nodes denote model elements, e.g., relations and attributes. Each such element is identified by an object identifier (OID).

Morphisms are binary relations over two, possibly overlapping, sets of OIDs. A morphism is typically used to represent a mapping between different kinds of models. Morphisms can always be inverted and composed.

Selectors are sets of node identifiers from a single or multiple models. These are denoted by S . A selector can be viewed as a relation with a single attribute, $S(V : OID)$, where V is a unique key.

Rondo offers operators for generating the alignments, composing them and applying them as data transformations. They are implemented upon these conceptual structures. Match is implemented in Rondo by using the Similarity flooding algorithm (Sect. 6.2.1).

Another system, called Moda, is described in (Melnik et al. 2005) in which correspondences are expressed as logical formulas. This system is more expressive than Rondo. Other model management systems include GeRoMe (Sect. 8.1.30), ModelGen (Atzeni et al. 2005, 2006) and MISM (Atzeni et al. 2009).

10.3.2 COMA++ (*University of Leipzig*)

COMA++ (Do and Rahm 2002; Do 2005) is a schema matching infrastructure built on top of COMA (Sect. 8.1.12). It provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for evaluating the effectiveness of the different matchers. COMA++ enables importing, storing and editing schemas (or models). It also allows various operations on the alignments among which compose, merge and compare. Finally, alignments can be applied to schemas for transforming or merging them.

Contrary to Rondo (Sect. 10.3.1), the matching operation is not described as atomic but rather described as a workflow that can be graphically edited and processed. Users can control the execution of the workflow in a stepwise manner and dynamically change execution parameters. The possibility of performing iterations in the matching process assumes interaction with users who approve obtained

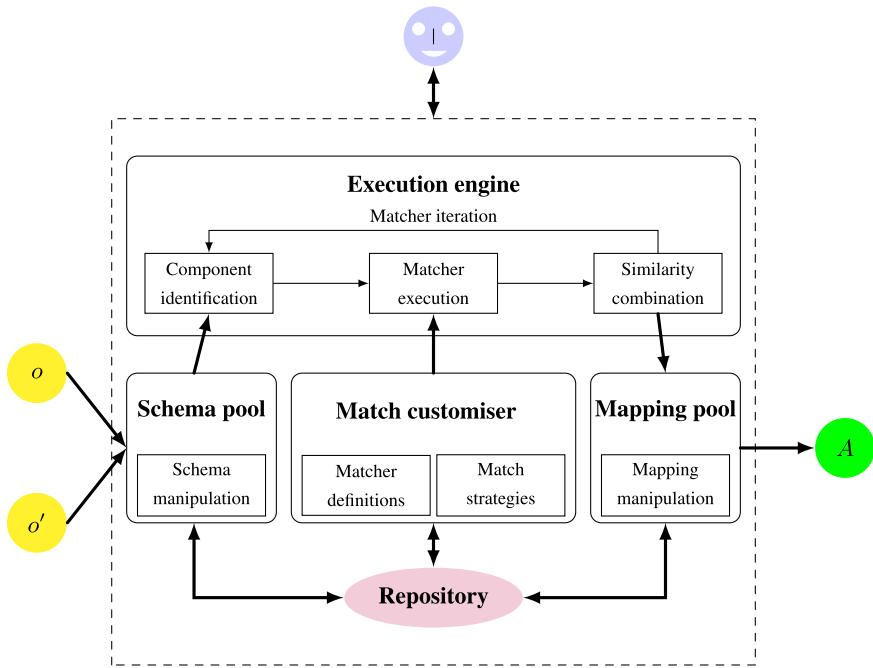


Fig. 10.3 COMA++ architecture (adapted from (Do 2005)).

matches and mismatches to gradually refine and improve the accuracy of the alignment (see Fig. 10.3). The matching operation is performed by the Execution engine based on the settings provided by the Match customiser, including matchers to be used and match strategies.

Data structures are defined in a homogeneous proprietary format. The Schema pool provides various functions to import and export schemas and ontologies and save them to and from the internal repository. Similarly, the Mapping pool provides functions to manipulate mappings. COMA++ can also export and import the matching workflows as executable scripts (similar to those manipulated in Rondo).

Finally, according to (Do 2005), there are other tools built on top of COMA++. For example, the CMC system provides a new weighting strategy to automatically combine multiple matchers (Tu and Yu 2005), while the work of (Dragut and Lawrence 2004) has adapted COMA to compute correspondences between schemas by performing a composition of the correspondences between individual schemas and a reference ontology. COMA++ was also integrated with an automatic target-driven merging system, thereby allowing for the semi-automatic generation of alignments to be further used for merging taxonomies from source to target as implemented in ATOM (Raunich and Rahm 2011).

10.3.3 GOMMA (*University of Leipzig*)

GOMMA is an infrastructure for Generic Ontology Matching and mapping MAnagement targeting the evolution of life science ontologies and mappings (Kirsten et al. 2011). It is made up of three levels:

- *repository*, which is responsible for data management, such as handling versions of ontologies and mappings;
- *functional components*, operating over data, such as *match*, *diff* and *evolution*;
- *tools*, built on top of the previous levels, such as *Ontology Matcher*, used to find mappings, complex ontology diff (COntoDiff), used to detect basic and complex changes between different ontology versions, or *ontology evolution explorer* (OnEx), used to explore and visualise the changes.

The matching approach was inspired by that of COMA++ (Sect. 10.3.2). GOMMA also supports parallel execution of matchers on multiple computing nodes or CPU cores as well as ontology partitioning (Sect. 7.1.1) to handle large-scale ontologies.

10.3.4 MAFRA (*Instituto Politecnico do Porto and University of Karlsruhe*)

MAFRA (MAppling FRAmework, already mentioned for its format in Sect. 10.1.1) is an interactive, incremental and dynamic framework for mapping distributed ontologies (Pinto da Silva 2004; Mädche et al. 2002). The framework consists of horizontal and vertical dimensions. The horizontal dimension covers mapping processes. It is organised according to the following components:

- *Lift and Normalisation*. This module handles syntactic, structural, and language heterogeneity. In particular, the lifting process includes translation of input ontologies into an internal knowledge representation formalism, which is RDF Schema. Normalisation (Sect. 5.2), in turn, includes (i) tokenisation of entities, (ii) elimination of stop words, and (iii) expansion of acronyms.
- *Similarity*. This module computes similarities between ontology entities by exploiting a combination of multiple matchers. First, *lexical similarity* between each entity from the source ontology and all entities from the target ontology is determined based on WordNet and altered Resnik measure (Sect. 5.2.2). Second, a *property similarity* is computed. This measures similarity between concepts based on how similar the properties they are involved in are. Finally, *bottom-up* and *top-down similarities* are computed. For example, bottom-up matchers take as input the property (dis)similarity and propagate it from lower parts of the ontology to the upper concepts, thus yielding an overall view of similarity between ontologies.

- *Semantic Bridging.* Based on the previously determined similarities, correspondences (bridges) between the entities of the source and target ontologies are established. Bridges, in turn, can be executed for the data translation task. The internals of bridges are discussed in detail in Sect. 10.1.1.
- *Execution.* The actual processing of bridges is performed in the execution module. This module translates instances from the source ontology to the target ontology. This translation can either be performed off-line, i.e., one time transformation, or on-line, i.e., dynamically, thus taking into account ‘fresh’ data, if there are.
- *Post-processing.* This module is in charge of the analysis and improvement of the transformation results, for instance, by recognising that two instances represent the same real-world object.

Components of the vertical dimension interact with horizontal modules during the whole mapping process. There are four vertical components. The *Evolution* module, in a user-assisted way, synchronises bridges obtained with the Semantic Bridging module according to the changes in the source and target ontologies. The *Cooperative Consensus Building* module helps users to select the correct mappings, when multiple mapping alternatives exist. The *Domain Constraints and Background Knowledge* module stores common and domain-specific knowledge, e.g., WordNet, precompiled domain thesauri, which are used to facilitate the similarity computation. Finally, a graphical user interface assists users in performing the mapping process with a desired quality.

10.3.5 The Protégé Prompt Suite (Stanford University)

Protégé is an ontology edition environment which offers design-time support for matching. In particular it features Prompt (Noy and Musen 2003), an environment that provides some matching methods and alignment visualisation. It is an interactive framework for comparing, matching, merging, maintaining versions, and translating between different knowledge representation formalisms (Noy and Musen 2003; Noy 2004b). The main tools of the suite include an ontology matching tool, called Anchor-Prompt (Sect. 8.1.9), an ontology-versioning tool, called PromptDiff, an interactive ontology merging tool, called iPrompt (Sect. 11.4.5; formerly known as Prompt), and a tool for factoring out semantically complete subontologies, called PromptFactor.

Prompt is implemented as an extension to the Protégé ontology editing environment. Thus, the Protégé browser provides overall capabilities for managing multiple ontologies. Prompt and Protégé are based on a frame-based knowledge model. Three types of frames are distinguished, namely classes, slots (properties) and instances. Because alignments are expressed in an ontology, they can be stored and shared through the Protégé server mode used in BioPortal (Sect. 12.7.1).

Below, we discuss only PromptDiff because Anchor-Prompt (Sect. 8.1.9) has been presented as a system and the interactive part of Prompt, iPrompt (Sect. 11.4.5) is presented in the user-interaction part.

PromptDiff (Noy and Musen 2002b) compares ontology versions and identifies the changes. This operation may be viewed as computing a Diff in model management (Sect. 10.3.1). PromptDiff produces a structural diff between two versions based on heuristics. It borrows many of them from iPrompt in order to identify what has changed from one version of an ontology to another. These heuristics include various techniques such as analysis and comparison of concept names, slots attached to concepts. The PromptDiff approach has two parts: (i) an extensible set of heuristic matchers; and (ii) a fixed point algorithm which combines the results of the matchers until they produce no more changes in the diff.

10.3.6 Alignment API and Implementation (INRIA)

A Java API (David et al. 2011) is available for manipulating alignments in the Alignment format and EDOAL. It defines a set of interfaces and a set of functions that they can perform.

Classes

The Alignment API is essentially made of four interfaces:

OntologyNetwork describes a network of ontologies through a set of ontologies and a set of alignments between them.

Alignment describes a particular alignment. It contains a specification of the alignment and a list of cells.

Cell describes a particular correspondence between entities.

Relation does not mandate any particular feature.

A full API to EDOAL for manipulating the EDOAL expressions has been provided with version 4.0.

To these interfaces implementing the Alignment format, are added a couple of other interfaces:

AlignmentProcess extends the Alignment interface by providing an align method. So this interface is used for implementing matching algorithms (**Alignment** can be used for representing and manipulating alignments independently of algorithms).

Evaluator describes the comparison of two alignments (the first one could serve as a reference). Each implemented measure must provide the eval method.

RendererVisitor provides a convenient way, using the visitor pattern, to traverse alignments.

An additional AlignmentException class specifies the kind of exceptions that are raised by alignment algorithms and can be used by alignment implementations.

Functions

The Alignment API provides support for manipulating alignments. As in (Bechhofer et al. 2003), these functions are separated from their implementation. It offers the following functions:

- Parsing and serialising** an alignment from a file in RDF/XML (`AlignmentParser.read()`, `Alignment.write()`);
- Computing** an alignment, with input alignment (`Alignment.align(Alignment, Properties)`);
- Thresholding** an alignment with threshold as argument (`Alignment.cut(double)`);
- Hardening** an alignment by considering that all correspondences whose strength is strictly greater than the argument are converted to \top , while the others are converted to \perp (`Alignment.harden(double)`);
- Comparing** one alignment with another (`Evaluator.eval(Properties)`) and serialising the result (`Evaluator.write()`);
- Outputting** alignments in a particular format, e.g., SWRL, OWL, XSLT, RDF, SPARQL, Silk LSL (`Alignment.render(visitor)`).

Matching and evaluation algorithms accept parameters. The parameters can be various weights used by some algorithms, some intermediate thresholds or the tolerance of some iterative algorithms. There is no restriction on the kind of parameters to be used.

The API implementation comes with:

- a complete and functional base implementation of the API;
- a command-line interface for running most of the API facilities;
- a library of sample matchers (not suitable for real work);
- a library of renderers (for RDF, XSLT, SWRL, OWL, C-OWL, SPARQL, SKOS, HTML);
- a library of evaluators implementing most of the measures of Sect. 9.3.1 as well as plotters like precision/recall curves or ROC curves;
- a library of wrappers for several ontology APIs (OWL-API, Jena, or SKOS) called Ontowrap.

The Alignment API also supports a server so that it is possible to store and share alignments on the web (Sect. 12.7.2). The Alignment API has been implemented in Java. This implementation has been used for various purposes: on-line alignment (Zhdanova and Shvaiko 2006) and Evaluation tool in the Ontology Alignment Evaluation Initiative (Sect. 9.1.3). In addition, many extensions use it for implementing matching algorithms, such as oMap (Sect. 8.3.4), CIDER (Sect. 8.1.35), and OLA (Sect. 8.3.8).

10.3.7 FOAM (*University of Karlsruhe*)

FOAM (Ehrig 2007) is a general tool for processing similarity-based ontology matching. It follows a general process which is presented in Fig. 10.4. It is made of the following steps:

- Feature engineering** selects the features of the ontologies that will be used for comparing the entities;
- Search step selection** selects the pairs of elements from both ontologies that will be compared;
- Similarity computation** computes the similarity between the selected pairs using the selected features;
- Similarity aggregation** combines the similarities obtained as the result of the previous step for each pair of entities;
- Interpretation** extracts an alignment from the computed similarity;
- Iteration** iterates this process, if necessary, taking advantage of the current computation.

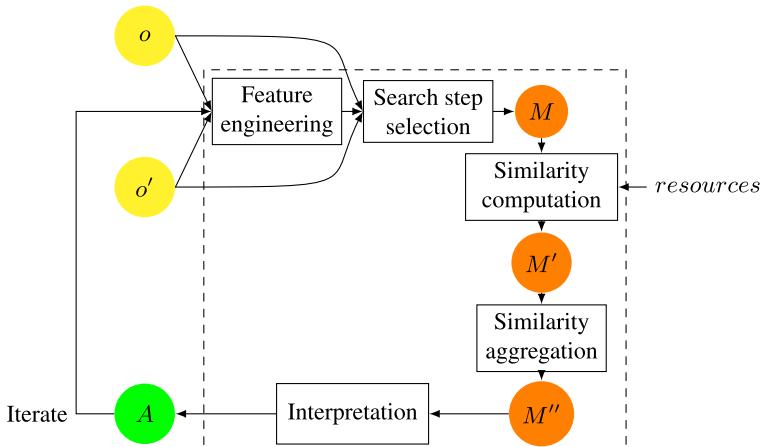


Fig. 10.4 FOAM architecture (adapted from (Ehrig 2007)).

The FOAM framework bundles several algorithms and strategies developed by its authors. Within this framework, matching systems such as NOM, QOM (Sect. 8.3.3), and APFEL (Sect. 8.4.1) have been cast. More systems can be integrated simply by changing any of the modules above. The global behaviour of the system can be parameterised through different scenarios, e.g., data integration, ontology merging, ontology evolution, query rewriting and reasoning, which offer default parameters adapted to these tasks.

FOAM itself is based on the KAON2 (Oberle et al. 2004) suite of tools and accepts ontologies in the OWL-DLP fragment. It offers a web-based interface. It is

also available as a Protégé plug-in. Finally, it offers translation tools from and to the Alignment format (Sect. 10.1.5) and other formats.

10.3.8 Harmony (*The MITRE Corporation*)

Harmony (Mork et al. 2008; Smith et al. 2009) is a schema integration workbench in which multiple tools share a common RDF-based knowledge repository—called an integration blackboard. It evolved into a collaborative effort, called open information integration, OpenII, which assembles a suite of open-source tools for information integration, e.g., matching, mediated schema creation and data exchange (Seligman et al. 2010; Smith et al. 2011).

It aims at facilitating the interoperation of research prototypes for matching that discover alignments with commercial matching tools that help produce instance transformations. The approach is based on a task model, which includes thirteen integration tasks that can be grouped into five phases as follows:

- schema preparation, i.e., identifying and preparing source and target schemas to be matched (this includes gathering available documentation about these schemas);
- schema matching, i.e., producing a set of correspondences;
- schema mapping, i.e., producing a set of explicit logical mappings (this includes developing domain, attribute, entity transformations, determining object identity, creating logical, e.g., LAV, mappings, verifying mappings against target schema);
- instance integration, corresponding to the execution of the generated transformation, i.e., linking instance elements and cleaning data, including reconciliation of remaining discrepancies;
- system implementation and deployment of the result.

The system handles XML and entity–relationship schemas. It combines various matching algorithms with a graphical user interface for manipulating the identified correspondences.

The confidence scores are shown graphically as colour-coded lines for inspection by users. The graphic user interface supports a variety of filters, e.g., thresholds for confidence scores, and whether a correspondence is human-established or machine-suggested, to help integration engineers focus attention on the desired correspondences. Finally, the iterative refinement of an alignment is supported through user feedback that accepts or rejects correspondences or marks fragments of schemas as completely addressed.

The workbench includes Schemr, a schema search engine that helps users discover and visualise relevant schemas in information integration tasks through sharing and reuse (Chen et al. 2011).

10.3.9 The NeOn Toolkit Alignment Plug-in

The NeOn Toolkit for ontology management features run-time and design-time ontology alignment support. It is based on the Alignment API (Sect. 10.3.6) and server (Sect. 12.7.2). The alignment plug-in (Euzenat and Le Duc 2012) allows for automatically computing and managing ontology alignments. More precisely, it offers the following functions:

- Retrieving alignments between ontologies or those available on the server,
- Matching ontologies,
- Trimming alignments by applying thresholds to existing alignments,
- Rendering alignments in a particular format,
- Uploading and storing alignments permanently on the server.

The NeOn Toolkit Alignment plug-in works in two modes: an off-line mode in which users can work locally on the alignments and an on-line mode in which it is connected to an Alignment server. Users can run the matchers which are embedded in the toolkit against ontologies in the NeOn Toolkit and manipulate alignments which are in the local environment.

The on-line mode connects the NeOn Toolkit to an Alignment server to share ontologies and to apply the same operations as above on alignments stored on the server. Of course, alignments can move back and forth between the server and the local environment.

Both on-line and off-line modes provide the functions of the Alignment API: retrieving alignments, matching ontologies, trimming alignments under various thresholds, storing them in permanent stores, and rendering them in numerous output formats. These operations support the whole alignment life cycle (Fig. 3.2, p. 58).

10.4 Summary

When considering ontology matching, one must decide how the resulting alignment will be used and delivered as well as how it will be produced. In particular, alignment formats allow for exchanging alignments among tools in well documented formats. They guarantee the openness of the result and everyone can take advantage of this result.

Moreover, the use of the alignment may raise constraints against the alignment to be used. As a consequence, matching systems should be able to qualify alignments as much as possible. This is what alignment metadata offers.

Some frameworks offer more than just a format, such as the ability to manipulate alignments and sometimes to apply them to data. The drawback of these environments, with the exception of the Alignment API, is that they cannot be easily embedded within other applications.

The choice to be made is dependent on the matching purpose of the matching tool. If its goal is to establish alignments that will be used in unknown and multiple

contexts, then choosing a format as neutral as possible is certainly the solution. If the matching result is used for processing some already known data, following a precise process, then a framework allowing this manipulation can be used. Finally, if the matching algorithm is to be integrated in a user driven environment, then an ontology and alignment editing tool is a good choice.

The two next chapters develop these themes by considering first user involvement in the matching process (Chap. 11) and then by discussing how alignments can be processed in applications (Chap. 12).

Chapter 11

User Involvement

This chapter considers how ontology matching techniques and human users interact from a technical perspective (rather than from organisational or social ones (von Hippel 2005)). This may occur because some functions of matching, such as finding anchors, are partially or completely carried out by individual users (Sect. 11.1) or sets of users (Sect. 11.2). The ability to explain alignments to users is also an important factor in the success of user involvement (Sect. 11.3). Finally, special attention is paid to tools for communicating with users and, more particularly, to alignment visualisers and editors (Sect. 11.4).

11.1 Individual Matching

The availability of users must be taken into account when designing the architecture of a matching system. This was one of the requirements from some applications mentioned in Chap. 1 and an important factor in the choice of a matcher as indicated in Chap. 3. There are at least three areas in which users may be involved in a matching solution (see Fig. 11.1):

- by providing initial alignments to the system (before matching),
- by configuring (which includes strategy and parameter selection) and tuning the system, and
- by providing feedback to matchers (during or after the automatic matching process) in order for them to adapt their results.

These three aspects are considered next. One of the differences between them is that, in the case of providing input, the system acts as a master that prompts users, while in the other two cases, both the user and the system may decide to act. On the one hand, the first and the third areas may be viewed as coarse-grained interactions and a matcher is viewed as a black box. On the other hand, the second area represents fine-grained interactions and a matcher is no longer viewed as a black box, since its internals have to be understood in order to meaningfully configure and tune it.

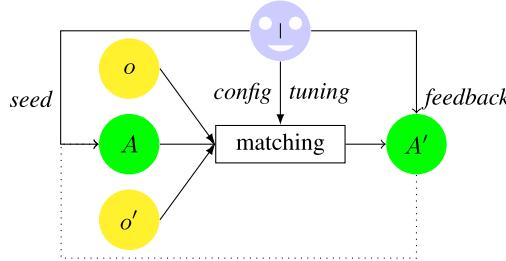


Fig. 11.1 General framework for individual user involvement. Users may provide a seed alignment A , which consists of some expected correspondences. Users also provide matcher configurations and may also *tune* matchers, by specifying method parameters and strategies used. Finally, users can provide *feedback* on resulting alignments which can be used by matchers, either to compute from a new seed alignment or to learn from user reaction.

11.1.1 Providing Input

This is the user task to provide input to matching systems. This obviously covers providing the ontologies to be matched.

More difficult is choosing the system parameters, which always depend on the method. Some algorithms may provide advice based on a priori analysis of data, but these techniques are specific to the matchers themselves.

Another important input that users can provide is an initial alignment which may be used as anchor by a global matching method. An initial alignment will constrain the matching system to produce an alignment complying with the initial alignment. This is an opportunity for users to control the system behaviour.

11.1.2 Manual Matcher Composition

We distinguish three ways to compose matchers:

Built-in composition corresponds to most of the systems presented in Chap. 8: the methods are composed according to the principles presented in Sect. 7.2. This composition is part of the algorithm and is applied to any data set given to the system.

Opportunistic composition corresponds to a system, that chooses the next method to run with respect to the input data and/or currently achieved results. Systems like Falcon-AO (Sect. 8.3.9), H-Match (Sect. 8.1.7) or RiMOM (Sect. 8.3.10) feature a limited implementation of this by choosing dynamically to use a particular matcher or not.

User-driven composition can consider the previous two options and is often used in environments where users can apply many different methods following their needs, such as Rondo (Sect. 10.3.1), COMA++ (Sect. 10.3.2), or Prompt (Sect. 10.3.5).

For instance, Fig. 11.2 shows the graphic user interface of COMA++ with a dialog for configuring its matching strategy for the two schemas under consideration. In turn, Fig. 11.3 shows that of AMC (Sect. 8.4.9), which provides visual support for designing matching processes, specifying parameters of their components (a), iterating the process (b), inspecting (c) and visualising (d) the intermediate results.

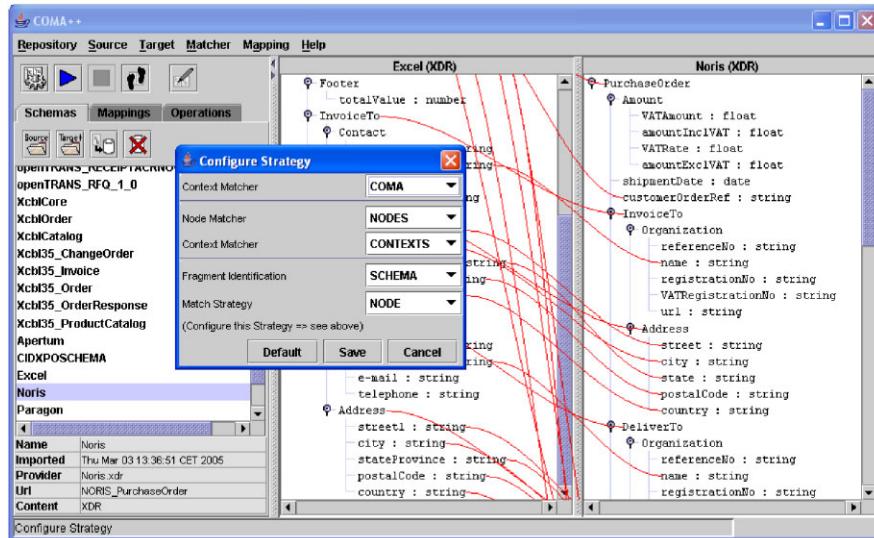


Fig. 11.2 Graphic user interface of COMA++ (Aumüller et al. 2005).

As another example, a system that may involve users in the dynamic assembly of matchers as follows. An interface could provide a library of basic matchers, filters and aggregators, as presented in Sect. 7.2, that users can assemble through graphic interaction. Then, the output of these methods could be materialised so that users can inspect them. Inspection can involve actually applying the resulting alignment for existing data in order to see the effects. Users may also dynamically change the parameters and quickly see the effects of these changes on the data. Finally it should be possible to save the designed architecture in order to use it in different applications.

11.1.3 Relevance Feedback

User feedback for each alignment or each specific correspondence found by a matcher may be used for improving the results. This may be achieved by adapting the local matcher parameters, such as

- matcher weights (Sect. 7.4.1),

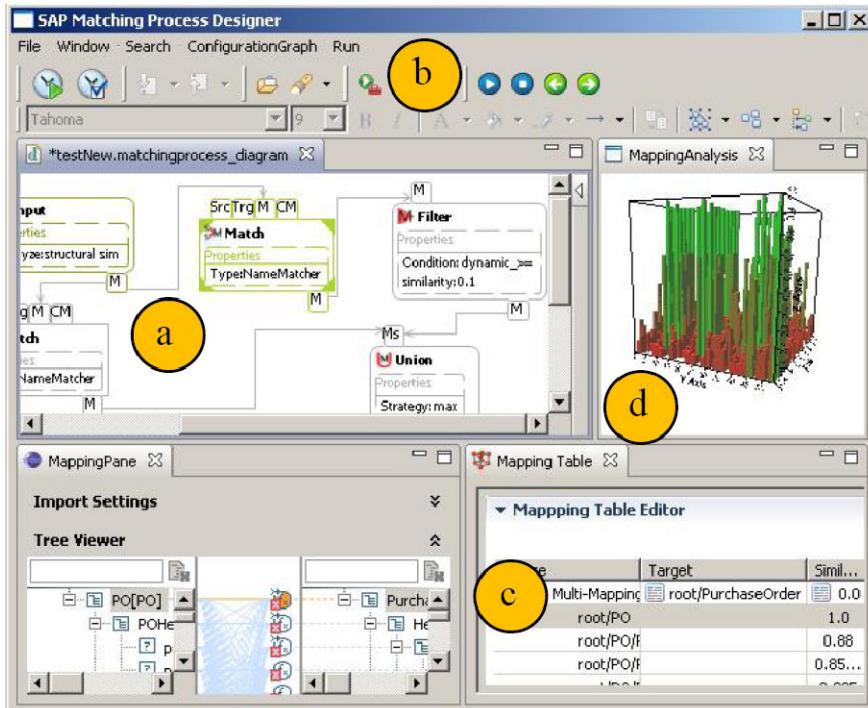


Fig. 11.3 Matching process designer of AMC (Peukert et al. 2011).

- thresholds used for filtering the results (Sect. 7.7), and
- aggregation parameters (Sect. 7.4).

Usually, systems compute a distance between the feedback and the result provided by the system. This is called the error. The computation of the error depends directly on the kind of feedback which is provided: users discarding unwanted correspondences, adding new correspondences or modifying the proposed correspondences.

Once the error is computed, the system has to select the parameter values that would minimise this error. This can sometimes be done directly: for instance, when the only parameter to set is a threshold, the system can directly compute a threshold that will provide the minimal error. However, very often such a method does not exist and it is necessary to use an indirect method that estimates the error reduction given some parameter value changes and searches for the best combination. Most of the methods presented in Sect. 7.6 may be used for this task. This kind of algorithm is used, for instance, in APFEL (Sect. 8.4.1). In turn, it is possible to implement a negative feedback method in which users tell if a correspondence is incorrect (Pachêco et al. 2011). This then affects the confidence in the correspondence (which is set to 0) and in future runs this will also affect the final confidence of correspondences across similar entities (by aggregating the confidences of all similar matches). (Lambrix and Kaliyaperumal 2013) proposed a session-based framework,

in which correspondences are delivered for user validation before the final alignment has been computed; this is particularly relevant for large-scale ontologies, since validating a large alignment at once is infeasible. Hence, correspondences can be partially validated and this process can be resumed later. Relevance feedback may also be used to improve alignments (Sect. 7.8). This is the case if the choice of the diagnosis to apply in debugging is delegated to users (Sect. 7.8.2).

A more active approach may be used in query-based applications. It asks users for a partial alignment with respect to terms involved in their queries. This can be achieved relatively easily by displaying the query and the target ontologies. Users who want to see their queries answered may show more willingness to help the system in this specific way, which can be viewed as an application of the ‘pay-as-you-go’ principle (Sarma et al. 2008).

We call ‘implicit matching’ serendipitously contributing to improve available alignments, i.e., users provide information about alignments by doing another task that matters to them without consciously dealing with alignments. Some systems use query logs to enhance match candidate generation (Sects. 8.1.36 and 8.3.18). Preliminary experiments for evaluating the characteristics of doing matching tasks embedded within user daily work have been run (Conroy et al. 2009). Users were asked to perform simple sessions of matching related to their interests and were rewarded (giving them interesting news feeds). The conclusion of the experiment is that as long as the task remains within user interests, it is bearable, and it would be better accepted if accomplished in the context of a relevant task. Such resulting local and partial alignments could be recorded as a special type of alignment that can be further reused when these alignments are necessary. A system could collect many of these alignment fragments, provided by many users over time, and the aggregation of many such partial alignments will eventually provide high-quality alignments between ontologies. Even when partial, such alignments may be sufficient for answering a large proportion of queries. Hence, local query alignment is a promising way to obtain alignments from users without asking them for a full alignment.

11.2 Collective Matching

Besides involving a single user at a time, mostly in a synchronous fashion, matching may also be a collective effort in which several users are involved (see Fig. 11.4). This would leverage the network effect. Indeed, if it is too cumbersome for one person to come up with a correct alignment between several pairs of ontologies, this can be more easily resolved by many people together, because (i) each person has to do a very small amount of work, (ii) each person can improve on what has been done by others, and (iii) errors remain in minority.

Collective approaches to matching, since they involve multiple parties in the process, allow for negotiating or correcting mismatches and mistakes collaboratively. Some steps in this direction have already been taken within the consensus building sessions of the Ontology Matching workshops (Caracciolo et al. 2008). This has

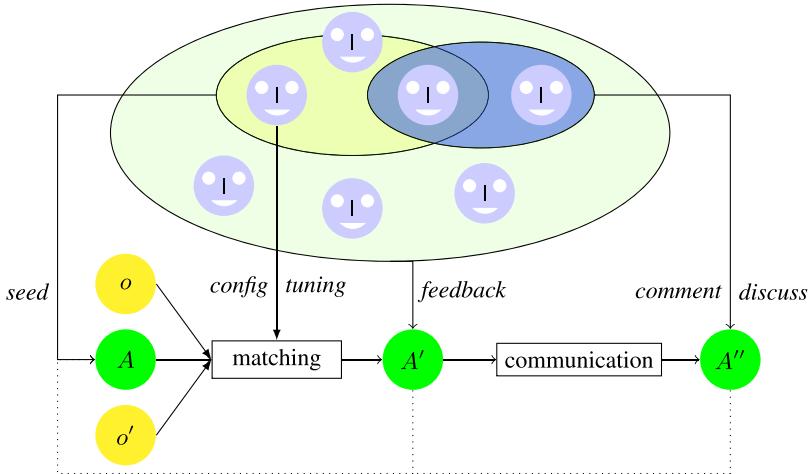


Fig. 11.4 General framework for collective matching. In complement to the individual actions (Fig. 11.1), multiple users, belonging to different communities of interest, e.g., biology, chemistry, illustrated by ellipses, may also *comment* and *discuss* correspondences which will influence the matcher behaviour.

also been considered in the field of multiagent systems where raw alignments are refined by agent negotiation (Sect. 7.4.3). We describe below several proposals for achieving collective matching among human users.

11.2.1 Community-Driven Ontology Matching

Community-driven ontology matching extends the notion of ontology matching by allowing a community of people to share alignments over the web, reuse them as well as argue about them by using annotations. Early experiments have been reported in (Zhdanova and Shvaiko 2006). Technically this requires, among other things, extending correspondences with metadata (Sect. 10.2) covering the correspondence author name, the application domain, and his or her trust value in a community.

The OntoMediate project (Correndo et al. 2008) proposed a system offering storage for ontologies and alignments, social interaction and collaboration. To that extent, the system allows users to annotate and vote in favour of or against proposed correspondences. All this social information is logged in the system in order to be exploited if necessary, for instance, to explain the arguments (see Sect. 11.3).

(Dewaraja 2010) reports an 18 question survey among 32 ontology matching users highlighting the needs for user support to matching. The resulting CLONTY system proposed a game, where two players are presented synchronously with the same pair of concepts. They independently assign possible relationships (equivalence, disjointness, hypernym, hyponym) to the pair of concepts. They gain points

if they reach an agreement on the relationship, which will be the output of the game. The process is supported by a web-based interface providing information and context about the two concepts. The game ends when ten concept pairs have been considered. Games with a purpose, engaging distributed groups of users, have been explored in the context of ontology matching as well, e.g., SpotTheLink (Thaler et al. 2011).

11.2.2 Crowdsourcing Ontology Matching

(McCann et al. 2008) proposed an application of mechanical turk to enlist the multitude of users in a community to help matching schemas by asking them simple questions, e.g., is monthly-fee-rate of type DATE?, and then learning from the answers to improve matching accuracy, e.g., in case of a positive answer to the preceding question above by using a specific *date* matcher. On the one hand, the questions should be relatively easy for users to answer and, on the other hand, they must affect substantially the matching accuracy. In this vein, three types of questions were used for three different purposes: (i) to verify intermediate predictions of the system, (ii) to learn domain integrity constraints, and (iii) to verify final match candidates. Users were classified into *trusted* and *untrusted*, based on their answers to a set of (evaluation) questions with answers known in advance. The answers from trusted users were further combined using a voting scheme. Ultimately, two user participation schemes were analysed: a standard *volunteering* scheme and a scheme in which users had to ‘*pay*’ by answering first several questions in order to use a desired service.

CrowdMap is an approach to acquire class correspondences through microtask crowdsourcing (Sarasua et al. 2012). It decomposes the matching problem into a set of smaller subproblems or *microtasks*, and publishes these on a crowdsourcing platform which submits them to on-line workers. They are expected to have neither specific domain skills, nor additional motivation for such tasks, other than a monetary reward, e.g., one to five cents. Each task may be carried out by several such workers and, based on the retrieved answers, the resulting alignment is computed, for instance by majority voting (Sect. 7.4.2). CrowdMap takes as input candidate correspondences, which are verified to improve their accuracy. The generated microtasks are of two types: (i) validation, in which the crowd is asked to specify if the displayed correspondence is correct, and (ii) identification, in which the crowd is asked to identify a particular relation, e.g., subsumption, between the classes at hand. These types of microtasks are presented through two specific user interfaces, e.g., in the first case it shows two classes under consideration together with the supporting contextual information, e.g., definitions, and poses a question if one concept is *the same as* another concept, which requires a yes/no answer. Verification questions (with correct answers known in advance) were used to reduce spam answers.

Collective matching requires alignment sharing. These systems identified the need for several capabilities, such as storage for the created alignments, recording

of the arguments and votes, and notification to entities and alignments of interest. Alignment services (Sect. 12.7) usually support sharing alignments or correspondences and their metadata. Some of them offer annotation editing and discussion lists as support for collaborative matching. Besides technology, the success of social and collaborative matching techniques will largely depend on the creation of a critical mass of users that actually use them, similarly to what happens to any type of data on the web.

11.3 Explaining Alignments

Matching systems produce alignments that may not be intuitively obvious to human users. In order for users to trust the alignments, and thus use them, they need information about them, e.g., they need access to the sources that were used to determine semantic correspondences between ontology entities. Explanations are also useful when matching large applications with thousands of entities, e.g., business product classifications, such as UNSPSC and eCl@ss. In such cases, automatic matching solutions will find many plausible correspondences, and hence user input is required for cleaning up the alignment. Finally, explanations can also be viewed and applied as argumentation schemas for negotiating alignments between agents.

There are only a few matching systems able to provide explanations for their results (Shvaiko et al. 2005; Dhamankar et al. 2004; Trojahn et al. 2011), though development of this topic may take advantage of progress in other areas, such as OWL justifications (Horridge et al. 2011, 2012). The solutions proposed so far focus on default explanations, explaining basic matchers, explaining the matching process, and negotiating alignments by argumentation.

In this section, we describe how a matching system can explain its answers, thus making the matching result mode intelligible. We first present the information required for providing explanations of matching and alignments (Sect. 10.2.2). Then, we discuss approaches to explanations of matching by examples of existing systems (Sect. 11.3.1). Details of these approaches are provided in the remainder, including default explanations (Sect. 11.3.2), explaining the basic matchers (Sect. 11.3.3), and explaining the matching process (Sect. 11.3.4).

11.3.1 *Explanation Approaches*

The goal of explanations is to take advantage of information about alignments, and in particular metadata (Sect. 10.2), for rendering the matching process intelligible to users. A key issue is to represent explanations in a simple and clear way (Léger et al. 2005).

In fact, while knowledge provenance and process traces may be enough for experts when they attempt to understand why a correspondence was returned, usually

they are inadequate for ordinary users. Thus, raw justifications have to be transformed into an understandable explanation for each of the correspondences, e.g., by showing general reasons for an incorrect match together with an example illustrating the reason, such as in mSeer (Sect. 8.4.5). Techniques are required for transforming raw justifications and rewriting them into abstractions that produce the foundation for what is presented to users. Presentation support also needs to be provided for users to better understand explanations. Human users will need help in asking questions and obtaining answers of a manageable size. Additionally, agents may even need some control over requests, such as the ability to break large process traces into appropriate size portions. Requirements for process presentation may include (McGuinness and Pinheiro da Silva 2004):

- methods for breaking up process traces into manageable pieces,
- methods for pruning process traces and explanations to help users find relevant information,
- methods for explanation navigation, including the ability to ask follow-up questions,
- methods for obtaining alternative justifications for answers,
- different presentation formats, e.g., natural language, graphs, and associated translation techniques,
- methods for obtaining justifications for conflicting answers,
- abstraction techniques.

There are several approaches to provide explanations of matching results. We describe below three such approaches. There are, however, few work on the topic in the literature and even fewer implemented systems. So, this chapter more specifically describes the explanation approaches as implemented in two systems, namely S-Match (Sect. 8.1.18) and iMAP (Sect. 8.2.6).

The Proof Presentation Approach

Semantic matchers usually produce formal proofs of their inferences as the basis for a correspondence. They can thus benefit from work developed for displaying and explaining proofs.

For instance, S-Match (Shvaiko et al. 2005) has been extended to use the Inference Web infrastructure as well as the Proof Markup Language (McGuinness and Pinheiro da Silva 2003; Pinheiro da Silva et al. 2006). Thus, meaningful fragments of S-Match proofs can be loaded on demand. Users can browse an entire proof or they can restrict their view and refer only to specific, relevant parts of proofs. The proof elements are also connected to information about basic matchers that generated the hypotheses.

The Strategic Flow Approach

Many matchers are composed of other matchers and have to decide in favour of some particular results over others. This composition and decision flow can be recorded in a dependency graph and used for providing explanation to users.

For instance, iMAP (Dhamankar et al. 2004) records dependencies at a very precise level (correspondence per correspondence) and can provide users with justifications for (i) existing correspondences, (ii) absent correspondences, and (iii) correspondence ranking. It provides explanations by extracting in the dependency graph the part that has an influence on the choice of a correspondence and generates an explanation in English from this extracted subgraph.

The Argumentation Approach

The argumentation approach considers the justifications or arguments in favour or against specific correspondences (Sect. 7.4.3). Argumentation theories can determine, from a set of arguments, the correspondences which will be considered to hold and those which will not.

Argumentation can be applied to justify matching results to users on the basis of arguments and counter-arguments or to negotiate the correspondences that should be in an alignment. So far, this approach has mainly been applied to agents negotiating alignments (Laera et al. 2006) rather than for explaining them.

The argumentative approach is different from the proof presentation approach, because it does not follow the formal proof of the correspondences. It is also more suitable when no such a proof exists.

11.3.2 A Default Explanation

A default explanation of alignments should be a short, natural language, high-level explanation without any technical details. It is designed to be intuitive and understandable by ordinary users.

The S-Match Example

We concentrate on class matching and motivate the problem by the simple catalogue matching example shown in Fig. 11.5. Assume that an agent wants to exchange or to search for documents with another agent. The documents of both agents are stored in catalogues according to class hierarchies o and o' , respectively. S-Match takes as input these hierarchies, decomposes the tree matching problem into a set of node matching problems, which are, in turn, translated into a propositional validity problem that can then be efficiently resolved using a sound and complete SAT solver (Sect. 6.5.1).



Fig. 11.5 Simple catalogue matching problem.

From the example in Fig. 11.5, trying to prove that the node with label Europe in o (denoted by Europe) is equivalent to the node with label Pictures in o' (denoted by Pictures'), requires constructing the following formula (see Sect. 6.5.1 for details of formula construction):

$$\underbrace{((\text{Images} \equiv \text{Pictures}') \wedge (\text{Europe} \equiv \text{Europe}'))}_{\text{Axioms}} \Rightarrow \underbrace{((\text{Images} \wedge \text{Europe}) \equiv (\text{Europe}' \wedge \text{Pictures}'))}_{\text{Context}_c \qquad \text{Context}_{c'}}$$

In this example, the negated formula is unsatisfiable, thus the equivalence relation holds between the nodes under consideration.

Assume that agent o' is interested in knowing why S-Match suggested a set of documents stored under the node with label Europe in o as the result to the query—‘find European pictures’. A default explanation is presented in Fig. 11.6. To simplify the presentation, whenever it is clear from the context to which classification a label under consideration belongs to, we do not tag it with the prime symbol (').

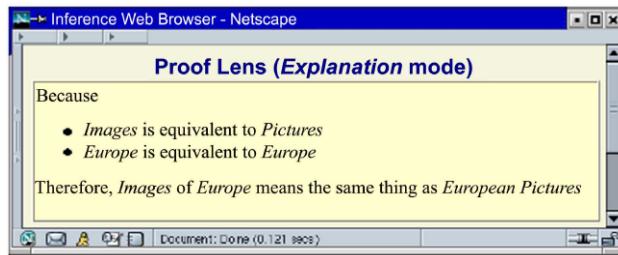


Fig. 11.6 S-Match explanation in English.

From the explanation in Fig. 11.6, users may learn that Images in o and Pictures in o' can be interchanged, in the context of the query. Users may also learn that Europe in o denotes the same concept as Europe in o' . Therefore, they can conclude that Images of Europe means the same thing as European Pictures.

The iMAP Example

iMAP (Sect. 8.2.6) differs substantially from S-Match. It is based on a combination of constraint- and instance-based basic matchers. Once the matcher has produced candidate correspondences, a *similarity estimator* computes, for each candidate, its similarity score. Finally, by applying the *match selector*, the best matches are returned as the final alignment.

Consider how iMAP explains why `pname = last-name` is ranked higher than `concat(first-name, last-name)`. Figure 11.7 shows the explanation as produced by iMAP (Dhamankar et al. 2004).

```
iMAP:
(1) Searcher Level:
text searcher generated last-name
text searcher generated concat(first-name,last-name)
text searcher ranked concat(first-name,last-name) higher by 0.102
(2) Similarity Estimator Level:
Name-based evaluator ranked last-name higher by 0.0419
Naive Bayes evaluator ranked concat(first-name,last-name) higher by 0.01
Final score for last-name: 0.434
Final score for concat(first-name,last-name): 0.420
(3) Match Selector Level:
Match selector did not modify two candidates
(4) Greatest influence on top five candidates for pname: Name-based evaluator
```

Fig. 11.7 iMAP explanation in English.

At the matcher level, `concat(first-name, last-name)` was ranked higher than the element with label `last-name`. It also clearly shows that things went wrong at the similarity estimator level. The naive Bayes evaluator still ranked matches correctly, but the name-based evaluator flipped the ranking, which was the cause of the ranking mistake.

The last line of the explanation also confirmed the above conclusion, since it states that the name-based evaluator has the greatest influence on the top five match candidates for `pname`. Thus, the main reason for the incorrect ranking for `pname` appears to be that the name-based evaluator has too much influence. This explanation would allow users to tune the system, possibly by reducing the weight of the name-based evaluator in the score combination step.

Users may not be satisfied with this level of explanations. Let us therefore discuss how they can investigate the details of the matching process by exploiting more detailed explanations, which are discussed in the forthcoming sections.

An Argumentation Example

Following Example 7.18 of Sect. 7.4.3, consensus alignment can also be achieved by a dialogue between agents during which they exchange arguments. Such a dialogue

is presented below. Agent C starts the dialogue by asserting alignment A between the two ontologies o and o' (the agent C is committed to support the alignment A and each correspondence it contains). A possible dialogue between C and P is as follows:

```
// Agent C is committed to support the alignment
C-assert( :content A :reply-with 1 )→ P
// Agent P asks to justify the correspondence  $\gamma_1$ 
(P does not have counter-a rgument)
C ←question( :content  $\gamma_1$  :reply-with 2 )—P
// Agent C justifies the correspondence  $\gamma_1$  with the arguments  $a_1$  and  $a_2$ 
C-support( :content  $a_1, a_2 \vdash^+ \gamma_1$  :in-reply-to 2 )→ P
// Agent P asks to justify the correspondence  $\gamma_3$  ( $P$  is ready to justify the opposite)
C ←challenge( :content  $\gamma_3$  :reply-with 3 )—P
// Agent C justifies the correspondence  $\gamma_3$  with the arguments  $a_3$  and  $a_4$ 
C-support( :content  $a_3, a_4 \vdash^+ \gamma_3$  :in-reply-to 3 )→ P
// Agent P contests the correspondence  $\gamma_3$  with the counter-arguments  $a_5$  and  $a_6$ 
C ←contest( :content  $a_5, a_6 \vdash^- \gamma_3$  :in-reply-to 3 )—P
// Agent C retracts the correspondence  $\gamma_3$ 
C-retract( :content  $\gamma_3$  :in-reply-to 3 )→ P
```

This results in the selection of the alignment $A' = \{\gamma_1, \gamma_2\}$.

In this dialogue, one of the agents may be a human user. The system knowing the user preferences can provide more adapted arguments.

11.3.3 Explaining Basic Matchers

Explaining basic matchers requires only to formulate the justification information. This is illustrated through S-Match.

Assume that an agent wants to see the sources of background knowledge used in order to determine a correspondence. For example, which applications, publications, other sources, have been used to determine that `Images` is equivalent to `Pictures`. Figure 11.8 presents the source metadata for the default explanation of Fig. 11.6.

In this case, both (all) the ground sentences used in the S-Match proof came from WordNet. Using WordNet, S-Match learnt that the first sense of the word `Pictures` is synonym to the second sense of the word `Images`. Therefore, S-Match can conclude that these two words are equivalent words in the context of the answer (Sect. 5.2.2). The meta-information about WordNet is also presented in Fig. 11.8 as *sources of the ground axioms*. Further examples of explanations include providing meta information about the S-Match library of element-level matchers, i.e., those which are based not only on WordNet, or the order in which matchers are used. This use of metadata is not restricted to S-Match and can be applied to any resource used in matching.

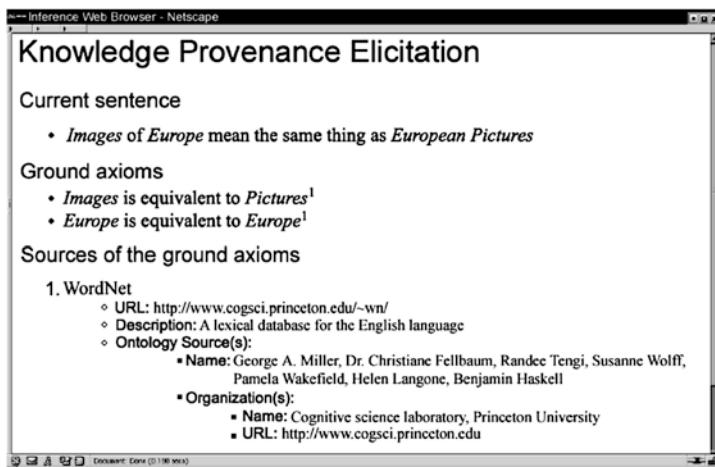


Fig. 11.8 S-Match source metadata information.

11.3.4 Explaining the Matching Process

S-Match and iMAP follow different matching strategies. iMAP follows a learning-based solution, while S-Match reduces the matching problem to a propositional validity problem. Let us discuss how they explain the matching process.

Dependency Graphs

Explanations of alignments in the iMAP system are based on the idea of a *dependency graph*, which traces the matchers, memorising relevant slices of the graph used to determine a particular correspondence. Finally, exploiting the dependency graph, explanations are presented to users as shown in Fig. 11.7.

The dependency graph is constructed during the matching process. It records the flow of matches, data and assumptions into and out of system components. The nodes of the graph are schema attributes, assumptions made by system components, candidate correspondences, etc. Two nodes in the graph are connected by a directed edge if one of them is the successor of the other in the decision process. Edges are labelled with the name of the system component that was responsible for the decision.

Figure 11.9 shows a dependency graph fragment that records the creation and flow for the correspondence *month-posted* = *monthly-fee-rate*. The *preprocessor* finds that both *month-posted* and *monthly-fee-rate* have values between 1 and 12 and hence makes the assumptions that they represent months. The date matcher takes these assumptions and generates *month-posted* = *monthly-fee-rate* as a candidate correspondence. This candidate is then scored by the name-based evaluator and the naive Bayes evaluator. The scores are merged by a *combining module* to produce a single

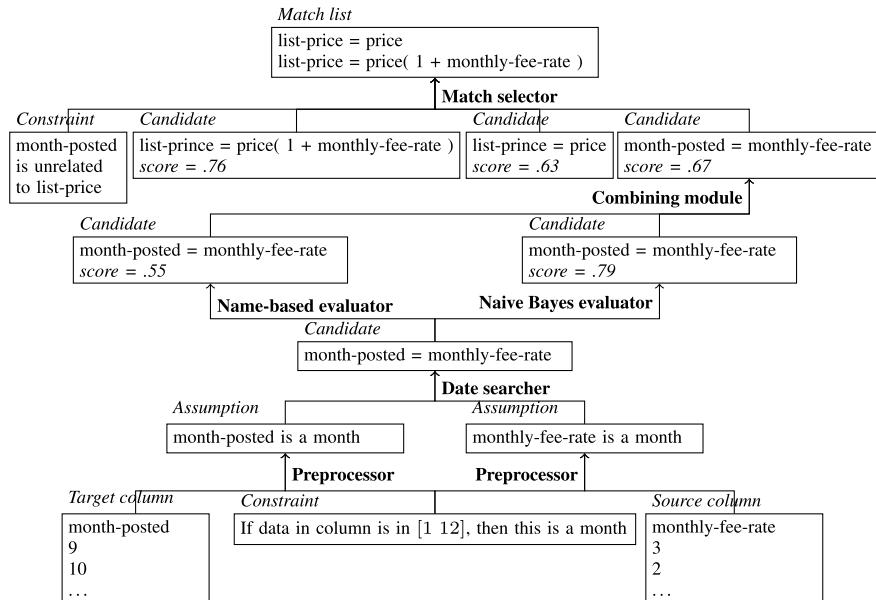


Fig. 11.9 Dependency graph as generated by iMAP (Dhamankar et al. 2004).

score. The *match selector* acts upon the several alignment candidates generated to produce the final list of alignments. Here, for the target attribute `list-price`, the selector reduces the rank of the candidate correspondence `price * (1 + monthly-fee-rate)` since it discovers that `monthly-fee-rate` maps to `month-posted`.

In each case, the system synthesises an explanation in English for users. To provide explanations, iMAP selects the relevant slices of dependency graph that record the creation and processing of a particular correspondence. For example, the slice for `month-posted = monthly-fee-rate` is the portion of the graph where the nodes participated in the process of creating that correspondence.

Explaining Logical Reasoning

A complex explanation may be required if users are not familiar with or do not trust the inference engine(s) embedded in a matching system. As the web starts to rely more on information manipulations, instead of simply information retrieval, explanations of embedded manipulations or inference engines become more important. In the current version of S-Match, a propositional satisfiability engine is used (Sect. 8.1.18), more precisely, this is the Davis–Putnam–Longemann–Loveland procedure (Davis and Putnam 1960; Davis et al. 1962) as implemented in JSAT/SAT4J (Le Berre and Parrain 2010).

The task of a SAT solver is to find an assignment ν of Boolean values $\{\top, \perp\}$ for atoms of a propositional formula φ such that φ evaluates to \top . φ is *satisfiable*

if and only if $\models v(\varphi)$ for some v . If there is no such an assignment, φ is *unsatisfiable*. A *literal* is a propositional atom or its negation. A *clause* is a disjunction of one or more literals. φ is said to be in conjunctive normal form if and only if it is a conjunction of disjunctions of literals. The basic DPLL procedure recursively implements three rules: *unit resolution*, *pure literal* and *split*. We only consider the unit resolution rule to simplify the presentation.

Let l be a literal and φ a propositional formula in conjunctive normal form. A clause is called a *unit clause* if and only if it has exactly one unassigned literal. *Unit resolution* is an application of *resolution* to a unit clause.

$$\text{unit resolution : } \frac{\varphi \wedge \{l\}}{\varphi[l \mid \top]}$$

Consider the propositional formula standing for the problem of testing if the concept with label Europe in o is less general than the concept with label Pictures in o' in Fig. 11.5. The propositional formula encoding the above stated matching problem is as follows:

$$\begin{aligned} ((\text{Images} \equiv \text{Pictures}) \wedge (\text{Europe} \equiv \text{Europe})) &\Rightarrow \\ ((\text{Images} \wedge \text{Europe}) &\Rightarrow (\text{Europe} \wedge \text{Pictures})) \end{aligned}$$

Its intuitive reading is ‘Assuming that Images and Pictures denote the same concept, is there any situation such that the concept Images of Europe is less general than the concept European Pictures?’’. The proof of the fact that this is not the case is shown in Fig. 11.10. Since the DPLL procedure of JSAT/SAT4J only handles conjunctive

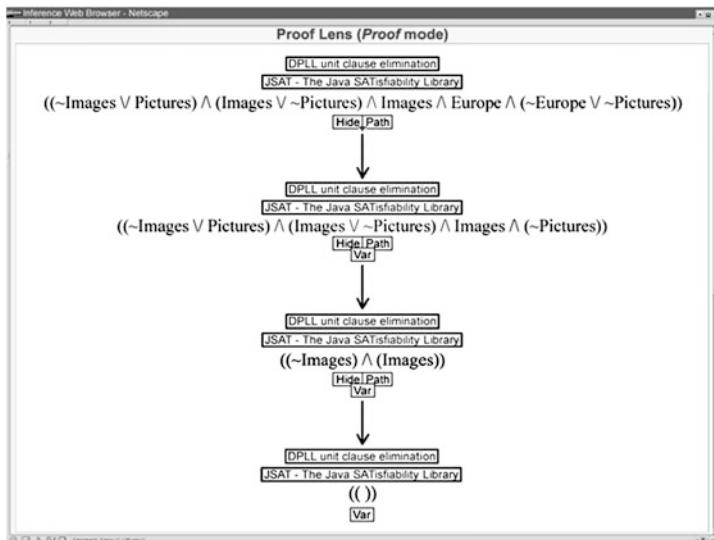


Fig. 11.10 A graphical explanation of the unit clause rule.

normal form formulas, in Fig. 11.10, we show the conjunctive normal form of the above formula.

From the explanation in Fig. 11.10, users may learn that the proof of the fact that the concept labelled Europe in o is less general than the concept labelled Pictures in o' requires 4 steps and at each proof step (excepting the first one, which is a problem statement) the *unit resolution* rule is applied. Moreover, users may learn the assumptions that are made by JSAT/SAT4J. For example, at the second step, the DPLL procedure assigns the truth value to all instances of the atom Europe, therefore making an assumption that there is a model where what an agent says about Europe is always true. According to the *unit resolution* rule, the atom Europe should be deleted from the input sentence. Thus it does not appear in the sentence of Step 2.

The explanation of Fig. 11.10 represents some technical details (only the less generality test) of the default explanation in Fig. 11.6. This type of explanations is the most verbose. It assumes that, even if the graphical representation of a decision tree is quite intuitive, users have some background knowledge in logics and SAT. However, if they do not, they have a possibility to learn it by following the publications mentioned in the source metadata information of the DPLL (see Sect. 10.2) *unit resolution* rule and JSAT, by clicking the *DPLL unit clause elimination* and the *JSAT-The Java SATisfiability Library* buttons, respectively.

By using explanations, a matching system can provide users with meaningful prompts and suggestions on further steps towards the production of a desired result. Having understood the alignments returned by systems, users can deliberately edit them manually, thereby providing feedback to the system. Besides explanations, matching systems should provide facilities for users to explore paths not followed by the system. These systems should enable users to re-launch the matching process with different parameters in an intermediate state.

11.4 Alignment Editors and Visualisers

The minimal requirement for interacting with users is to present them the objects on which they have to work, i.e., alignments. Once an alignment has been discovered, it should be presented and, if necessary, explained, such that users can interpret it and give feedback on it. Presenting and manipulating alignments under the formats provided in Sect. 10.1 may not be acceptable for many users. Hence, alignment editors and visualisers should offer human users the opportunity to be involved in the matching process. Below, we enumerate requirements for alignment editors and visualisers which draw on individual (Sect. 11.1) and collective (Sect. 11.2) matching as well as on alignment explanations (Sect. 11.3):

- It should be possible to have an overview of an alignment, such that users can understand which fragments of ontologies match and which do not. This should allow for focussing on specific areas of interest to be further examined.
- It should be possible to search, browse, zoom on specific correspondences as well as filter and sort them, based on different aspects, such as confidence values or their status, e.g., automatically generated, approved by a human user.

- For each generated correspondence, it should be possible to request an explanation (at different levels of detail) for why it was generated.
- Users should have visual facilities allowing them to accept, reject, create, update, delete, and postpone for later revision the correspondences under consideration.
- In a collaborative setting, when multiple users work on alignments, they should have immediate opportunity to comment, annotate or vote on the correspondences under consideration.
- In a crowdsourcing setting, it should be possible to assign fragments of ontologies to be outsourced to communities of interest as well as to load the respective (even partial) results once these become available.
- Systems should be able to influence the search for the desired alignment on various levels via unified interfaces, e.g., by suggesting sources of background knowledge to use, by supporting adequate matcher configuration and tuning, or through critique visualisation of intermediate results, thus stimulating users to become active and conscious in their choices. Ideally, each design-time function should be available from an alignment editor.

We describe a few systems for presenting and sometimes editing alignments. They sometimes offer interesting features, besides the mere representation of ontologies and alignments.

In general, such tools typically provide *line*- or *list*-based representation of alignments and *tree* and/or *graph*-based representation of ontologies. Colors are often used to represent different types of correspondences. Interaction is typically synchronised between ontologies and alignments. Most of the software developed for editing alignments are rather design tools. Such systems provide a convenient display of the currently edited alignments and the opportunity to discard, modify or add correspondences. Since ontologies and alignments may be very large, it is challenging to offer intuitive alignment editing support.

We present below editors and visualisers by first considering standalone software specifically dedicated to alignments, before considering software integrated within ontology management environments.

11.4.1 WSMT (DERI, University of Innsbruck)

The Web Service Modeling Toolkit (WSMT) (Kerrigan et al. 2007) is an integrated development environment (IDE) for semantic web services with ontology engineering capabilities. WSMT provides a set of tools for manually creating, editing and storing ontology alignments. WSMT and the ontology engineer work together in an iterative process. This involves cycles consisting of suggestions from the tool side and validation or creation of correspondences from user side. It offers a set of methods that assist ontology engineers in their work, such as different graphical perspectives over the ontologies, suggestions of the most related entities from the source and the target ontology, and guidance throughout the matching process (Mocan and

Cimpian 2007). Specifically, the suggestion algorithm implements a weighted average of the lexical and structural similarity factors, called eligibility factor. Pairs of entities with the eligibility factor higher than a threshold are retained as suggestions. Design-time graphical instruments support a formal model for alignment creation. Through this model the explicit links between graphical elements and correspondences are captured, and thus, the complexity of the underlying logical languages is hidden. Users provide graphic input, which is then translated into a logical formalism to be executed in run-time data mediation. WSMT is based on the idea that particular graphical visualisations, called *perspectives*, of ontologies to be matched or switching between combinations of perspectives can facilitate discovery of certain types of correspondences. Several types of perspectives were introduced. For example, the *partOf* perspective is the most common one and is used to focus on correspondences between concepts and attributes; while the *instanceOf* perspective focusses on instances and can be used to create conditional correspondences, e.g., by restricting the allowed values for the source or the target attributes. Graphically only one nesting level is displayed in order to facilitate guidance through the matching process.

11.4.2 Muse (University of California, University of Toronto)

Muse (Mapping Understanding and deSign by Example) is a mapping design wizard that uses data examples to assist integration engineers in designing mappings of a desired specification (Alexe et al. 2008a). The intuition behind it is that integration engineers usually understand better their data than alignments. The algorithms behind Muse build on two particular aspects of mapping design: (i) specification of the desired grouping semantics for sets of data, (ii) disambiguation of candidate correspondences possessing several alternative interpretations, namely when there is more than one way to obtain an atomic target schema element. The background idea of these algorithms is to infer the desired semantics based on the yes/no answers provided by users to a series of automatically generated questions exploiting small examples, possibly familiar to the database users. This should help in exemplifying to users (with familiar data examples) the nuances of how small changes to a correspondence change its semantics. Muse is also able to identify cases when a source instance is insufficient for illustrating all design alternatives, and hence, artificial examples are generated to illustrate these alternatives.

11.4.3 iMerge (Duisbourg University)

The iMerge editor (El Jerroudi and Ziegler 2008) has been developed for displaying matching results in the context of merging biomedical ontologies. It offers a classical tree view of the ontologies to be merged, but introduces an interesting matrix

view of the alignment, where the matrix dimensions are the two ontologies. The interface allows for folding and unfolding the ontologies, and thus, for concentrating on particular areas of the ontologies. It also offers a colour representation of confidence measures (see Fig. 11.11).

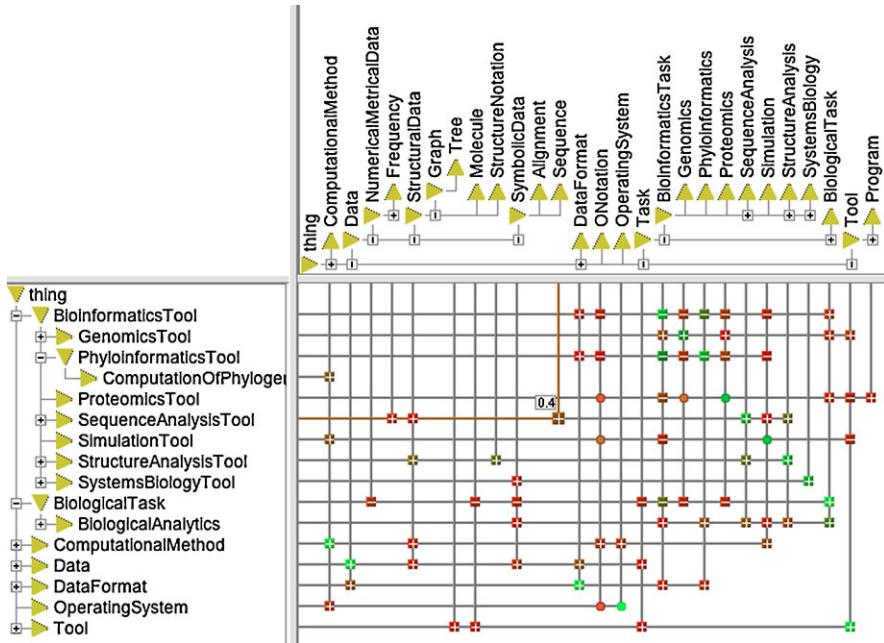


Fig. 11.11 iMerge interface presenting alignments. The ontologies are displayed as unfoldable trees, which unveil correspondences displayed in different colours depending on their status and confidence (that are displayed when the mouse hovers on a node).

11.4.4 Chimaera (Stanford University)

Chimaera is a browser-based environment for editing, merging and testing (diagnosing) large ontologies (McGuinness et al. 2000). Users are provided with a graphical user interface (the Ontolingua ontology editor) for editing taxonomies and properties. The interface contains spring-loaded menus with about seventy commands, e.g., merge classes. There are also various diagnosis commands which provide support for tests and changes, e.g., tests for redundant superclasses, slot values or type mismatch. Matching in the system is performed as one of the subtasks of a merge operation. Chimaera searches for merging candidates as pairs of matching terms, with terminological resources such as term names, term definitions, possible acronym and expanded forms, names that appear as suffixes of other names. It generates name resolution lists that help users in the merging task by suggesting terms which

are candidates to be merged or to have taxonomic relationships not yet included in the merged ontology. The suggested candidates can be names of classes or slots. These names to be resolved are presented through the menu of items, showing the current one, with the possibility to delete it or to move to the previous or the next item. The result is output in OWL descriptions similar to those presented in the OWL format (Sect. 10.1). Chimaera also suggests taxonomy areas that are candidates for reorganisation. These edit points are identified by heuristics, e.g., looking for classes that have direct subclasses from more than one ontology.

11.4.5 iPrompt (Stanford University)

iPrompt (Noy and Musen 2000) is an interactive interface to the Protégé Prompt Suite (Sect. 10.3.5) integrated in the Protégé ontology edition environment. It takes as input two ontologies and leads users towards one merged ontology as output. First, iPrompt creates an initial set of matches based on lexical similarity between class names. Here any name-based technique (Sect. 5.2) may be used. Then, it proceeds through the following cycle:

- Users choose an operation to perform from the iPrompt suggestion list. Some examples of operations are: merge classes, slots, instances, perform a shallow (deep) copy of a class.
- iPrompt performs the operation chosen at the previous step. It also identifies inconsistencies, e.g., name conflicts, redundancy in the class hierarchy, that the operation introduced, as well as possible strategies to resolve them. Finally, it generates a list of suggestions (concerning the next actions) for users.

Some techniques, such as rearranging lists of suggestions, are used in order to keep users focussed on the most important aspects of the given state of the process, thus converging on a desired merged ontology efficiently.

11.4.6 AlViz (Vienna University of Technology, Norwegian University of Science and Technology)

AlViz is a multiview Protégé plug-in for visual ontology matching (Lanzenberger and Sampson 2006). It provides traditional J-Trees views as well as the *small world graphs* view (van Ham and van Wijk 2004) for representing ontologies. In the first visualisation type, trees have roots and expandable indented node hierarchies. However, with large ontologies, this view does not provide adequate overview functionality. The second visualisation type, small world graphs, represents an abstraction of an ontology graph via clusters. Specifically, graph nodes are clustered according to the selected (through sliders) level of detail, thereby offering detail-and-overview functionality (see Fig. 11.12).



Fig. 11.12 AlViz: visualisation of ontologies with different matching entities (Lanzenberger and Sampson 2006).

In this modality, nodes represent ontology entities connected through relations, such as *is-a*, *part-of*, which can be filtered. The size of the nodes depends on the number of clustered concepts. The approach scales up to ontologies with thousands of nodes. Six categories of correspondence relations are supported, corresponding to different colours. In particular: equality (entities shown in red), syntactic equality (entities shown in orange), broader- or narrower-than (entities shown in blue-violet), respectively, similar (entities shown in green), different (entities shown in yellow). Clusters of nodes inherit the colour of the underlying nodes. Thus, users can apprehend similar entities, dissimilar entities, etc. The system supports zooming as well as selecting and highlighting. For example, when an entity in one ontology is selected, the matched nodes in the other ontology are highlighted, thereby providing a synchronised navigation. The FOAM system (Sect. 10.3.7) was reused to generate candidate correspondences visualised in AlViz. Users can revisit them visually by assigning the desired type to a relation, thereby approving or rejecting them.

11.4.7 CogZ (University of Victoria)

CogZ (cognitive support and visualization for human-guided mapping systems) is a realisation of a cognitive support framework that aims at reducing the cognitive load

on users in ontology matching (Falconer and Storey 2007; Falconer et al. 2009). It was implemented as a plug-in for the Prompt suite (Sect. 10.3.5). The framework focusses on the relationship between users and matching tools. It approaches, beyond handling various matching algorithms implemented in Prompt, such issues as assisting users in remembering what correspondences they have looked at and executed, understanding results of an automatic matching algorithm, revising previously made decisions or making temporary decisions on correspondences.

CogZ supports incremental search and filtering of the ontologies to be matched as well as of the generated correspondences, thereby reducing the matching scope. This helps users focus on the desired fragments of the ontologies and hide distracting information from the screen. Ontology entities may have annotations added as superscripts to their labels, namely, the letter *c* means that the respective correspondences were discovered automatically, while the letter *m* means that these entities were validated and matched. When an ontology term is selected, its correspondences are highlighted, while all the other correspondences become semitransparent. Finally, CogZ provides ‘fisheye’ zoom that keeps at normal font size the selected entities from two ontologies, while other concepts are shown in progressively smaller fonts depending on their relevance to the selected entities.

11.5 Summary

Ontology matching can be performed by programs which cannot accept any user input. But it may also be performed in cooperation with users, especially when automatic solutions do not achieve desired results. This happens either through interactive matching sessions or through indirect feedback provided by using the produced alignments.

Articulating user interaction within the various ontology matching architectures discussed before is not an easy task. It requires research in its own right on designing the type of interaction that can occur, e.g., matcher initialisation, solicitations from matchers or post hoc judgement. It demands appropriate user interfaces for displaying matching results and offering intuitive action means. This observation is relevant for both design-time and run-time matching, in order to design interaction schemes which are burdenless to users. At design time, interaction should be both natural and complete; at run time, it should be hidden in the user task. Beyond a simple display, systems must offer (on demand, with different levels of detail) explanation facilities enabling users to fully understand matching results.

Moreover, collaborative participation of several people in the matching task has been investigated in the past few years. This is a natural trend that will certainly be reinforced and will require more support for both: systems stimulating users and users helping each other.

Such a system-user collaboration may be extended to the actual use of alignments in operational tasks, which are the object of the next chapter.

Chapter 12

Processing Alignments

In this book, we have taken a two-step view on reducing semantic heterogeneity: (i) matching of entities to determine alignment and (ii) processing the alignment according to application needs. In the previous chapters, we have discussed various themes related to the first step. In this chapter, in turn, we present how the alignments can be specifically used by applications, thus focussing on the alignment processing step.

Since this book is devoted to ontology matching, our goal is not to present a complete panorama of the different uses of alignments. This would require another book. Rather, we present the broad classes of alignment use and the tools for implementing these usages. Meanwhile, most of the commercially available ontology integration tools focus on automation of alignment processing, by opposition to matching. They are very often specialised in a particular segment of the matching space. Al-tova MapForce and Stylus Studio XSLT Mapper are specialised in XML integration. They integrate data from XML sources as well as databases or other structured sources. Microsoft BizTalk Schema Mapper targets business process and information integration, using the proprietary BizTalk language. Semafora SemanticIntegrator offers ontology-based integration of data coming from databases or ontologies. There are unfortunately no scholar references describing these systems in depth and URLs change so often that we refer the reader to ontologymatching.org for accurate and up to date information. Similarly, instance matchers, like Silk (Sect. 12.4.2), can be considered as tools processing alignments for generating links.

The matching operation itself is largely not automated within these tools, though they facilitate manual matching by visualising input ontologies (XML, database, flat files formats, etc.) and the correspondences between them. However, they have improved over the past years and have started including approximate terminological matching. Once the correspondences have been established it is possible to specify, for instance, some data translation operations over the correspondences such as adding, multiplying, and dividing field values in the source document and storing the result in a field in the target document.

We discuss below a minimal set of operations that can be performed from alignments, including ontology merging (Sect. 12.1), ontology transformation

(Sect. 12.2), data translation (Sect. 12.3), data interlinking (Sect. 12.4), mediation (Sect. 12.5), and reasoning (Sect. 12.6). They are presented as operators in the style of model management (Sect. 10.3.1). We conclude this chapter with issues related to alignment management (Euzenat et al. 2008a). In particular, available services for storing and sharing alignments are presented (Sect. 12.7), and the problem of updating alignments along with their evolution is discussed (Sect. 12.8).

12.1 Ontology Merging

Ontology merging is a first natural use of ontology matching. As depicted in Fig. 12.1, it consists of obtaining a new ontology o'' from two matched ontologies o and o' so that the matched entities in o and o' are related as prescribed by the alignment. If o , o' and $\text{Merge}(o, o', A)$ are expressed in the same language, merging can be presented as the following operator:

$$\text{Merge}(o, o', A) = o''$$

The ideal property of a merge would be that

$$\begin{aligned}\text{Merge}(o, o', A) &\models o \\ \text{Merge}(o, o', A) &\models o' \\ \text{Merge}(o, o', A) &\models \alpha(A)\end{aligned}$$

if $\alpha(A)$ is the alignment expressed in the logical language of $\text{Merge}(o, o', A)$, and

$$o, o', \alpha(A) \models \text{Merge}(o, o', A)$$

The former set of assertions means that the merge preserves the consequences of both ontologies and of the relations expressed by the alignment under the reduced semantics (Sect. 2.5.3). The latter assertion means that the merge does not entail more consequences than specified by the semantics of alignments (Sect. 2.5.3). Of course, this is not restricted to the union of the consequences of o , o' and A .

Example 12.1 (Generating OWL axioms) With the same ontologies as in Fig. 10.1, and the following correspondences:

$$\text{id} = \text{isbn} \quad \text{Book} \leq \text{Volume}$$

It is possible to generate OWL axioms as follows:

```
o:id owl:equivalentProperty o':isbn .
o:Book rdfs:subClassOf o':Volume .
```

More generally it is possible to generate complex OWL axioms from EDOAL alignments. For instance, the correspondence

```

<Cell>
<entity1>
  <edoal:AttributeOccurrenceRestriction>
    <edoal:onAttribute>
      <edoal:Relation rdf:about="&onto1;#hasWritten"/>
    </edoal:onAttribute>
    <edoal:comparator rdf:resource="&edoal>equals"/>
    <edoal:value>
      <edoal:Literal edoal:type="xsd:integer" edoal:string="1" />
    </edoal:value>
  </edoal:AttributeOccurrenceRestriction>
</entity1>
<entity2><edoal:Class rdf:about="&onto2;#Writer"/></entity2>
<relation></relation>
<measure rdf:datatype='xsd:float'>1.</measure>
</Cell>

```

can be rendered as the corresponding OWL axiom:

```

<owl:Class rdf:ID="&onto2;#Writer">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&onto1;#hasWritten"/>
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>

```

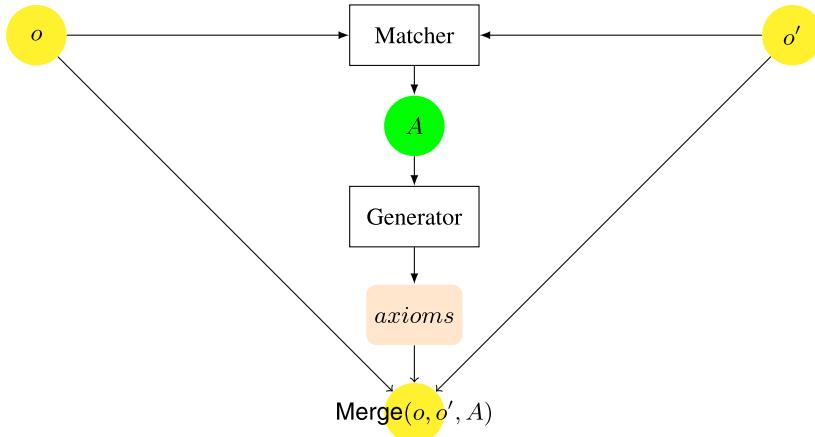


Fig. 12.1 Ontology merging. From two matched ontologies o and o' , resulting in alignment A , articulation $axioms$ are generated. This creates a new ontology covering the matched ontologies.

When ontologies are expressed in the same language, merging often involves putting the ontologies together and generating bridge or articulation axioms. Merging does not usually require a total alignment: those entities which have no corresponding entities in the other ontology will remain unchanged in the merged ontology. Ontology merging is especially used when it is necessary to carry out reasoning

involving several ontologies (see Sect. 12.6). It is also used when editing ontologies in order to create ontologies tailored for a particular application. In such a case, it is most of the time followed by a phase of ontology reengineering, e.g., suppressing unwanted parts from the obtained ontology.

Usage Examples Prompt (Sect. 10.3.5) and Rondo (Sect. 10.3.1) offer independent operators for ontology merging. OntoMerge (Sect. 12.1.1) takes bridge rules expressed in the predicate calculus and can merge ontologies in OWL. We discuss it in more details below. The Alignment API (Sect. 10.3.6) can generate axioms in OWL or SWRL for merging ontologies. Other systems are able to match ontologies and merge them directly: FCA-merge (Sect. 8.2.3), SKAT (Sect. 8.1.5), DIKE (Sect. 8.1.4). OntoBuilder (Sect. 8.1.10) use ontology merging as an internal operation: the system creates an ontology that is mapped to query forms. This ontology is merged with the global ontology so that queries can be directly answered from the global ontology.

12.1.1 *OntoMerge (Yale University and University of Oregon)*

OntoMerge (Dou et al. 2005) is a system for ontology translation on the semantic web. Ontology translation refers here to such tasks as (i) data set translation, i.e., translating a set of facts expressed in one ontology to another, (ii) generating ontology extensions, i.e., given two ontologies o and o' and an extension (subontology) o_s of the first one, build the corresponding extension o'_s , and (iii) query answering from multiple ontologies. The main idea of the approach is to perform ontology translation by ontology merging and automated reasoning. Input ontologies are translated from a source knowledge representation formalism, e.g., OWL, to an internal representation, which is Web-PDDL (McDermott and Dou 2002). Merging two ontologies is performed by taking the union of the axioms defining them. Bridge axioms or bridge rules are then added to relate the terms in one ontology to the terms in the other. Once the merged ontology is constructed, the ontology translation tasks can be performed fully automatically by mechanised reasoning. In particular, inferences are conducted either in a demand-driven (backward chaining) or data-driven (forward chaining) way depending on the task, with the help of a first-order theorem prover, called *OntoEngine*. It is assumed that bridge rules are provided by domain experts, or by other matching algorithms, which can discover and interpret them with a clear semantics. Finally, OntoMerge supports bridge rules which can be expressed using the full power of the predicate calculus.

12.2 Ontology Transformation

Ontology transformation, from an alignment A between two ontologies o and o' , generates an ontology o'' expressing the entities of o with respect to those of o'

according to the correspondences in A . It can be denoted by the following operator:

$$\text{Transform}(o, A) = o''$$

Contrary to merging, ontology transformation, and the operators to follow, are oriented. This means that the operation has an identified source and target and from an alignment it is possible to generate two different operations depending on source and target.

Ontology transformation is not well supported by tools. It is useful when one wants to express one ontology with regard to another one. This can be particularly useful for connecting an ontology to a common upper-level ontology, for instance, or local schemas to a global schema in data integration.

12.3 Data Translation

Data translation, presented in Fig. 12.2, translates instances from entities of ontology o into instances of connected entities of matched ontology o' . This may be expressed by the following operator:

$$\text{Translate}(d, A) = d'$$

Data translation usually involves generating a transformation program from the alignment.

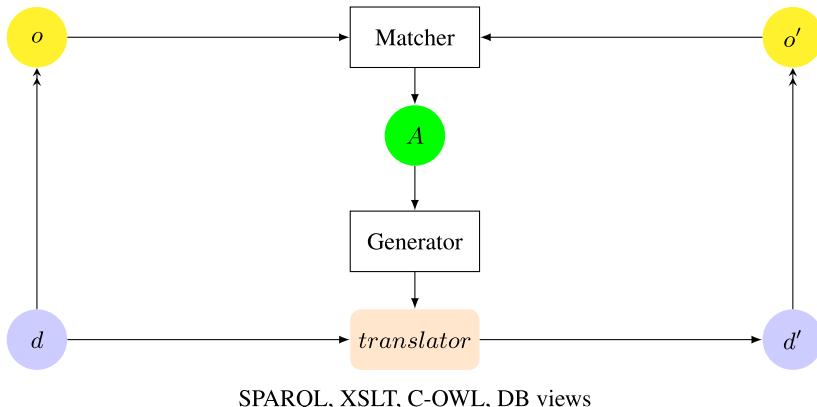


Fig. 12.2 Data translation. From two matched ontologies o and o' , resulting in alignment A , a *translator* is generated. This allows the translation of the instance data (d) of the first ontology into instance data (d') for the second one.

This requires a total alignment if one wants to translate all the extensional information. Partial alignments risk losing instance information in the translation (this can also be acceptable if one does not want to import all the instance information).

Data translation is used for importing data under another ontology without importing the ontology itself. This is typically what is performed by database views in data integration (Sect. 1.2), in multiagent communication for translating messages (Sect. 1.6.1), and in semantic web services for translating the flow of information in mediators (Sect. 1.5).

Data translation may be achieved using the SPARQL CONSTRUCT primitive (Euzenat et al. 2008c). From an alignment A , the process first generates a set of SPARQL queries, which are applied to the data, and then generate RDF graphs. Hence, the implemented mechanism is as follows:

$$\text{Translate}(d, A) = \text{generateCONSTRUCTQueries}(A)(d) = d'$$

SPARQL-based data translation is dependent on the alignment language. The first proposal is based on complex languages, such as EDOAL (Sect. 10.1.6), which are able to express the subtleties of the alignment and, in particular, the dependency between classes and properties and data transformations.

Example 12.2 (SPARQL CONSTRUCT generation from EDOAL correspondences) The correspondence of Fig. 10.1, which expresses that a Pocket book in the left-hand side ontology corresponds to a Volume, whose size is less than 14, in the right-hand side ontology, expressed in EDOAL as in Sect. 10.1.6, can be transformed in the following SPARQL query:

```
CONSTRUCT { ?x rdf:type o:Pocket . }
WHERE {
  { ?x rdf:type o':Volume.
    ?x o':size ?y. }
  FILTER ( ?y <= 14 )
}
```

Such a query transforms all the Volumes of small size into Pocket books. Moreover, if a transformation is attached to the correspondence for expressing size in centimeters in height in inches, units can be converted in the same way. However, it is only possible to express the converse translation partially: all Pocket books can be transformed into Volumes, but their size cannot be generally inferred (the transformation may be seen as weakening).

This first approach has the advantage of using expressive alignments, which can express very precise transformations. However, not all correspondences can be transformed into CONSTRUCT queries. For instance, disjunctive concepts cannot be inserted in the CONSTRUCT part.

Another proposal has been designed in the Mostro system (Rivero et al. 2011). It only deals with simple alignments matching named classes and properties (level 0 alignments). In order to offer meaningful transformations, it first identifies kernels for each correspondence. Kernels are groups of correspondences in the alignment (level 0 alignments) and concepts in the ontologies which are tied together by (ontological) constraints. From each kernel it then generates a CONSTRUCT SPARQL query by expressing the constraints as graph patterns.

Example 12.3 (SPARQL CONSTRUCT generation from level 0 correspondences)
Consider simple correspondences expressing that:

$$\text{topic} = \text{subject} \quad \text{Pocket} \leq \text{Volume}$$

on the same ontologies of Fig. 10.1. It is possible for the first correspondence to cluster Pocket with topic and Volume with subject in a pattern that links them together, and to generate the following query:

```
CONSTRUCT { ?x rdf:type o':Volume . ?x o':subject ?y . }
WHERE { ?x rdf:type o:Pocket. ?x o:topic ?y. }
```

The reverse query is not generally possible, since not all Volumes are Pocket books. However, it will still be possible to generate:

```
CONSTRUCT { ?x o:topic ?y . }
WHERE { ?x o':subject ?y . }
```

because it would still hold according to the second correspondence. In this case, it is not possible to perform data transformations.

One issue with these approaches is the order of query applications. Some of these queries may match several times the same entity and generate several assertions concerning them. In general, if entities are identified by URIs, the only problem is redundancy, but this still remains to be studied in depth.

Usage Examples Rondo (Sect. 10.3.1) provides tools for data translation. The Alignment API (Sect. 10.3.6) can generate translations in XSLT, SPARQL CONSTRUCT or C-OWL. Many tools developed for data integration can generate translators under the form of SQL queries. Drago (Sect. 12.6) can process alignments expressed in C-OWL for transferring data from one ontology to another one. Some of the tools reviewed in Chap. 8, provide their output as data translation or process themselves the translation. These include Clio, ToMAS (Sect. 12.8.1), TransScm (Sect. 8.1.3), MapOnto (Sect. 8.1.16) and sPLMap (Sect. 8.2.12). In addition, other systems, such as Spicy are specialised in data translation from alignments. We discuss Clio and Spicy in more details below.

12.3.1 Clio (IBM Almaden and University of Toronto)

Clio is a system for managing and facilitating data transformation and integration tasks within heterogeneous environments (Miller et al. 2000, 2001; Fagin et al. 2009). Clio handles relational and XML schemas. As a first step, the system transforms input schemas into an internal representation, which is a nested relational model. The Clio approach is focussed on making the alignment operational. It is assumed that the matching step, namely, identification of the *value correspondences*, is performed with the help of a schema matching component or manually.

The built-in schema matching algorithm of Clio combines in a sequential manner instance-based attribute classification via a variation of a naive Bayes classifier (Sect. 7.5.1) and string matching between elements names, e.g., by using an edit distance (Sect. 5.2.1). Then, taking the $n\text{-}m$ value correspondences (the alignment) together with constraints coming from the input schemas, Clio compiles these into an internal query graph representation. In particular, an interpretation of the input correspondences is given. Thus, a set of logical mappings with formal semantics is produced. To this end, Clio also supports mapping composition (Fagin et al. 2005). Finally, the query graph can be serialised into different query languages, e.g., SQL, XSLT, and XQuery, thus enabling actual data to be moved from a source to a target, or to answer queries. The system, besides trivial transformations, aims at discovering complex ones, such as the generation of keys, references and join conditions.

Starting from the mappings provided by Clio or other declarative mapping specifications, such as of IBM Rational Data Architect, the Orchid system was designed to convert such declarative schema mappings into data flow specifications of data warehousing environments, such as ETL (Extract Transform Load) jobs, and vice versa (Dessloch et al. 2008). It provided a common model into which both mappings and ETL jobs can be transformed as its instances, which can be further optimised and deployed into multiple target environments.

12.3.2 Spicy (University of Basilicata, ICAR-CNR)

Spicy (Bonifati et al. 2008), like its extensions +Spicy (Mecca et al. 2009) and ++Spicy (Marnette et al. 2011), is a system for generating data translation programs in SQL or XQuery and for verifying their quality. The system takes as input a source and target schema and an alignment made as a set of tuple-generating dependencies. The alignment may be provided in a file, generated by a matcher or specified through a graphical user interface. Spicy is able to generate transformations from the alignment and to evaluate the quality of transformations in the presence of existing data populating both schemas. Specifically, candidate mappings are checked and only those which represent better transformations of the source into the target are selected. A query induced by a correspondence is run on a subset of the source, the result of which is compared to the available target instance. This comparison helps in identifying incorrect transformations originating from wrong correspondences as well as in ranking (in order to reduce human intervention) the remaining match candidates to be suggested to users for inspection. The verification method compares the structure and actual content of data sources, viewed as trees. Spicy uses electrical circuits techniques to compare features, such as the topology and content of tree structures, in order to obtain quickly a measure for their similarity. Features can be aggregated through three alternative strategies: arithmetic mean, harmonic mean, and Euclidean distance.

12.4 Data Interlinking

Data interlinking aims at finding a set of links, mostly `owl:sameAs` statements, between two RDF data sets. Data interlinking can be considered independently from any ontology alignment (Ferrara et al. 2011b). However, it is sometimes possible to process alignments as linking specifications (Scharffe and Euzenat 2011). This can be expressed by the following operator:

$$\text{Interlink}(d, d', A) = L$$

in which an alignment A is used for generating a link set L between two data sources d and d' .

As in Sect. 12.3, it is possible to perform data interlinking by generating SPARQL CONSTRUCT queries, through reusing similar techniques.

Example 12.4 (Data interlinking based on SPARQL CONSTRUCT) With the same ontologies as in Fig. 10.1, and the correspondences

$$\text{id} = \text{isbn} \quad \text{Book} \leq \text{Volume}$$

it is possible to generate the following SPARQL query:

```
CONSTRUCT { ?x owl:sameAs ?a . }
WHERE
{ ?x rdf:type o':Volume .
  ?x o':isbn ?i .
  ?a rdf:type o:Book .
  ?a o:id ?i . }
```

Such a query generates `owl:sameAs` links for instances of `Volume` and `Book` with identical `isbn` and `id`. If correspondences are expressed in EDOAL with transformations between the property values, it is possible to take the transformations into account in the query.

Usage Examples Several tools are dedicated to generating links either from alignments, such as KnoFuss, or from linking specifications, which can be considered as alignments, such as Silk. We briefly present these below.

12.4.1 KnoFuss (*The Open University*)

KnoFuss (knowledge fusion) (Nikolov et al. 2008) aims at performing data-level interlinking of OWL knowledge bases. It assumes that the data to be linked is structured according to the same ontology. It provides a method selection mechanism to choose and configure individual matchers considering class hierarchies with the goal of (i) coreference resolution, (ii) conflict detection, and (iii) inconsistency resolution. These tasks are executed in sequence.

Each task is supported by a library of basic methods, and a generic workflow selects, executes and combines them, based on method metadata. For instance, a Jaro-Winkler (Sect. 5.2.1) label matcher is annotated as being appropriate for coreferencing with a default threshold of .87. However, based on the application context, a different fine-tuned configuration might be applied. Technically, *application contexts* are organised hierarchically and specify method parameters for different cases, e.g., publication matching or journal article matching. These values override the default values, e.g., by applying the threshold of .93 instead of .87.

KnoFuss has been extended by performing ontology matching in order to improve instance coreference resolution (Nikolov et al. 2009). It uses existing auxiliary instance-level links in order to enhance schema-level correspondences between classes, thus implementing context-based matching with linked data as background knowledge (Sect. 7.3). In particular, linked individuals are taken from different repositories, such as YAGO or Musicbrainz. For each individual, its respective classes in both ontologies are identified. Correspondences between classes are established based on an overlap similarity coefficient to quantify the subset of shared individuals, thereby distinguishing between strongly correlated classes and merely non-disjoint ones (Sect. 5.4.2). This is an example of the loop, illustrated in Fig. 1.5 (p. 12), between data interlinking and ontology matching reinforcing each other. The obtained alignment narrows down the search space by excluding from the comparison instances that are unlikely to be linked: this is a typical example of ontology matching used as blocking (Sect. 7.1.1). When there are several correspondence alternatives, the relation with a more generic class is preferred to that with a more specific one, because this may lead to additional coreference resolution links.

12.4.2 Silk (Chemnitz University of Technology, Freie Universität Berlin)

Silk (Volz et al. 2009) is a link discovery framework for the web of data. It generates RDF links between data items based on user-provided link specifications expressed using the Silk link specification language.

The Silk link discovery engine is responsible for the link discovery from a Silk specification. A Silk specification first identifies the data sources (through SPARQL endpoints or local files) and then contains interlink specifications. The Silk engine operates in the following phases following the interlink specification structure:

Restricting (`SourceDataset` and `TargetDataset`) identifies the cluster of individuals to be compared (Sect. 7.1.1). It thus performs the *blocking* part of the algorithm. These restrictions are implemented as simple SPARQL queries.

Comparison (`LinkageRule`) computes a similarity value for each pair of instances. The data items are loaded into an internal cache. Then, for each pair of data items a similarity value in [0 1] is computed, which is called link condition evaluation, and which stands for the confidence of the candidate link. The

link condition evaluation comprises an extensible set of comparison methods the results of which are combined using aggregation functions (Sect. 7.4).

Filtering (`Filter`) selects links in two steps. Initially, all the links with a confidence lower than a user-defined threshold are discarded. Then, all links that originate from the same subject are grouped together. It is possible to limit the number of links per subject, thereby favouring the selection of links with higher confidence.

Generating links (`Outputs`) outputs in a specific way (file, SPARQL endpoint) the links that pass the filtering phase using the `LinkType` specified in the script, e.g., `owl:sameAs`. Silk offers the possibility to use different thresholds to send to different outputs, so separating the links with very high confidence from those which must be checked by a user (Sect. 7.7.1).

The framework is offered in three variants: (i) Silk single machine, used to generate RDF links between two data sets on a single machine, (ii) Silk map-reduce, used to scale to large data sets by using multiple machines, and (iii) Silk server, used as an identity resolution or matching component. The Silk server is designed to take as input a stream of RDF instances, to match their descriptions to a local set of known instances, thereby discovering (missing) links between them. Incoming instances which do not match a known instance are added to the local set of instances continuously. It also provides a web interface that can be used to refine links or fine-tune linking specifications. Finally, LinQuer (Hassanzadeh et al. 2009) offers similar functions for relational databases with a less expressive language.

12.5 Mediation

In this section, we consider a mediator as an independent software component that is introduced between two other components in order to help them interoperate. There are many different forms of mediators, including some acting as brokers or dispatchers. We concentrate here on query mediators which can perform two operations:

$$\text{TransformQuery}(q, A) = q'$$

and

$$\text{Translate}(a', \text{Invert}(A)) = a$$

`TransformQuery` is a kind of ontology transformation which transforms a query expressed using ontology o into a query expressed with the corresponding entities of a matched ontology o' . The `Translate` operation performs data translation on the answer of the query, if necessary. This process is presented in Fig. 12.3.

Translating answers requires the possibility of inverting the alignments (`Invert` operator). The generated functions should be compatible, otherwise the translated answer may not be a valid answer to the initial query. Compatibility can be expressed as follows:

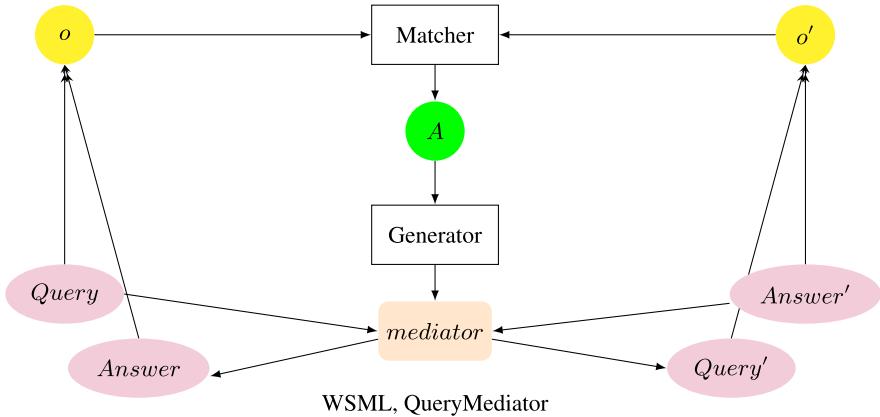


Fig. 12.3 Query mediation. From two matched ontologies o and o' , resulting in the alignment A , a *mediator* is generated. This allows for transforming queries expressed with the entities of the first ontology into a query using the corresponding entities of a matched ontology and the translation back of the results from the second ontology to the first one.

$$\forall e \in o, \quad \text{TransformQuery}(\text{TransformQuery}(e, A), \text{Invert}(A)) \sqsubseteq e$$

Here we use a subsumption relation (\sqsubseteq), but it can be replaced by any suitable relation ensuring that the answer is compatible. However, it is not always necessary to translate answers, since they can be objects independent from the ontologies, e.g., picture files, strings.

Example 12.5 (SPARQL CONSTRUCT generation from the EDOAL correspondence) Given the following query:

```
SELECT ?x ?i
FROM o
WHERE
{ ?x rdf:type o:Pocket .
?x o:id ?i .
?x o:topic "Politics" . }
```

with the alignment of Fig. 10.1 and the added correspondences:

$$\text{id} = \text{isbn} \quad \text{topic} \leq \text{subject}$$

it is possible to transform this query as follows:

```
SELECT ?x ?i
FROM o'
WHERE {
{ ?x rdf:type o':Volume .
?x o':isbn ?i .
?x o':size ?y .
?x o':subject "Politics". }
FILTER ( ?y <= 14 )
}
```

A more efficient transformation is:

```

SELECT ?x ?i
FROM o'
WHERE {
  ?x rdf:type o':Politics .
  ?x o':isbn ?i .
  ?x o':size ?y . }
  FILTER ( ?y <= 14 )
}

```

but it needs recognising complex patterns.

It is relatively easy to generate correct query transformations by recognising simple patterns in queries. It is more difficult to be complete, i.e., to ensure that all the patterns occurring in an alignment have been considered, due to the difficulty of recognising complex patterns. Moreover, as for data translation, there is a problem of confluence: since several correspondences may be used for transforming the same triple, is the result independent from their application order? Part of the problem can be solved by ordering correspondences along the nature of the matching elements (Makris et al. 2010), but in the general case, the problem remains open.

Usage Examples Query mediation is mainly used in data integration (Sect. 1.2) and peer-to-peer systems (Sect. 1.4). The Alignment API (Sect. 10.3.6) can behave as a SPARQL query mediator from simple alignments by straightforwardly substituting URIs. Some systems directly generate mediators after matching, such as Wise-Integrator (Sect. 8.3.6). When the mediator content is expressed as SQL view definitions, many database systems can process them. In turn, (Correndo et al. 2010) described SPARQL query rewriting with alignments expressed in a language where atomic formulas, typically triples, can be transformed into compound formulas, typically graph patterns, with the addition of functional dependencies expressing how to compute values of the latter from those of the former. This approach has been extended to deal with a subset of the EDOAL alignments. It also tackles the problem of dereferencing URIs appearing in the initial query, because there may be several URIs identifying the same individual. This is achieved by wrapping a dereferencing service¹ in functional dependencies. Similarly, (Makris et al. 2010) described a generation process for an expressive language (less expressive than EDOAL). It works triple by triple and differs depending on whether the predicate in the triple is in the RDF, RDFS or OWL vocabulary or not.

12.6 Reasoning

Reasoning consists of using alignments as rules for reasoning with the two matched ontologies. Bridge axioms used for merging can also be viewed as such rules.

$$\text{TransformAsRules}(A) = o$$

¹<http://sameas.org>.

Here the set of rules is represented as an ontology o which must be written in an ontology language supporting rules or the expression of bridge axioms.

Typically, any transformation of the alignments under a form suitable for reasoning, such as SWRL (Sect. 10.1.4) or OWL (Sect. 10.1.2) can be used by inference engines for these languages, such as Pellet (Sirin et al. 2007) or HermiT (Motik et al. 2009c). For instance, LogMap (Sect. 8.3.26) also offers similar facilities.

Usage Examples ALCOMO (Sect. 7.8.2), provides support for diagnosing and repairing inconsistent or unsatisfiable aligned ontologies. It uses standard OWL reasoners for that purpose and takes the Alignment format (Sect. 10.1.5) as input. It interprets the alignments with the reduced semantics (Sect. 2.5.3) and can only answer inconsistency queries.

Drago (Serafini and Tamilin 2005) is a distributed reasoning system in which each peer uses a collection of ontologies and oriented alignments, called mappings, across ontologies of different peers. It interprets such a system as a C-OWL network of ontologies following the DDL semantics (Sect. 2.5.3). The reasoner can answer consistency and satisfiability queries at one peer. It can also answer entailment queries over OWL axioms (not alignments). Reasoning is performed in a distributed manner by adding to the Pellet reasoner a ‘bridge expansion rule’, which interprets the mappings.

IDDL (Zimmermann and Le Duc 2008; Le Duc et al. 2010) interprets networks of ontologies under the equalising semantics (Sect. 2.5.3). It accepts ontologies in OWL and alignments in the Alignment format (Sect. 10.1.5). Reasoning can be local, by using the reduced semantics, or distributed by using the OWLLink protocol. Reasoning is performed by using any standard OWL reasoner.

12.7 Alignment Services and Repositories

There are several reasons why applications using ontology matching could benefit from sharing matching techniques and results:

- *Each application can benefit from more algorithms:* many different applications have comparable needs. It is thus appropriate to share the solutions to these problems. This is especially true as quality alignments are often quite difficult to provide.
- *Each algorithm can be used in more applications:* alignments can be used for different purposes and must be expressed as such instead of as bridge axioms, mediators or translation functions.
- *Each individual alignment can be reused by different applications:* there is no magic algorithm for quickly providing a useful alignment. Once high quality alignments have been established—either automatically or manually—it is very

important to be able to store, share and reuse them, in order not to lose that quality result.

For that purpose, it is useful to provide alignment services able to store, retrieve and manipulate existing alignments as well as to generate new alignments on-the-fly. Such services could be shared by applications using ontologies. They should be seen as a directory or a service by web services, as an agent by agents, as a library in ambient computing applications, etc. Alignment servers should provide the following functions:

- a way to manipulate alignments, i.e., loading, inspecting and rendering them;
- the ability to store alignments persistently and retrieve them;
- communication over the network for delivering required alignments;
- a proper annotation in order for the clients to evaluate the opportunity to use one of them or to start from them.

In addition they may provide:

- on-the-fly ontology matching: for automatically generating new alignments;
- alignment edition: for interactively updating alignments by adding or modifying correspondences;
- ontology integration: for navigating within ontologies and inspecting them;
- alignment processing: for processing the alignments to translate queries or to interlink data, for instance;
- alignment evaluation: for comparing different alignments using various methods.

These services can be used by applications as well as by the infrastructure itself. They can be used at two different moments in applications:

at design time through invocation by design and engineering environments: they can be integrated within development environments, where they will be loosely coupled components, that may be asked for providing alignments and for exploiting these alignments (like the NeOn Toolkit through the NeOn Alignment plug-in; Sect. 10.3.9).

at run time alignment servers can be invoked directly by the applications.

Such services require standardised support, such as the choice of an alignment format (Sect. 10.1) or at least of a metadata format (Sect. 10.2).

Usage Examples There have been several proposals for providing matching systems and alignment stores that can be considered as servers (Euzenat 2005; Zhdanova and Shvaiko 2006; d’Aquin and Lewen 2009), but they need a wider availability (to agents, services, etc.) and should reach a critical mass of users to really be helpful. We briefly present three systems, which have been developed with this perspective.

12.7.1 BioPortal (Stanford University)

BioPortal² is an open community-based repository of biomedical ontologies (Noy et al. 2008), though its underlying technology is domain-independent. It was built by using Protégé to store ontologies in various languages. It uses an ontology to store correspondences between named objects and associated metadata. All mappings are stored together since there is no notion of alignment. With the help of a web-based user interface, users can interactively create 1:1 mappings between concepts, upload and download mappings created with other tools, such as Prompt (Sect. 10.3.5), browse and discuss controversial mappings, and visualise them as well as their associated metadata, e.g., general comments on the mapping, users who created or uploaded it, or its creation date (Sect. 10.2). Finally, users can filter mappings based on different criteria, e.g., by user name or by source ontology. Filtered mappings can be downloaded. Similarly, applications can access filters and the filtered mappings through a web service.

BioPortal provides access to the ontologies and the correspondences in an integrated way (to the point that correspondences may be considered as ontology metadata (Noy et al. 2008)), allowing for more comprehensive explanations. It was bootstrapped with more than 30 000 mappings from seven sources, such as FMA, Mouse anatomy, NCI thesaurus, and Gene Ontology. These correspondences may have been obtained manually or automatically. (Ghazvinian et al. 2009) extended the BioPortal repository by creating more than four million concept mappings between 140 ontologies of BioPortal and 67 terminologies of UMLS (Sect. 2.1.2). A simple matching method was used based on string similarity of preferred terms and synonyms over normalised strings. Then, network analysis techniques were applied to identify hubs and clusters over the ontologies. Mappings were found useful to guide newcomers to the field to its most representative ontologies, to identify areas that are insufficiently covered, or to identify ontologies that might be used as background knowledge in specific matching tasks.

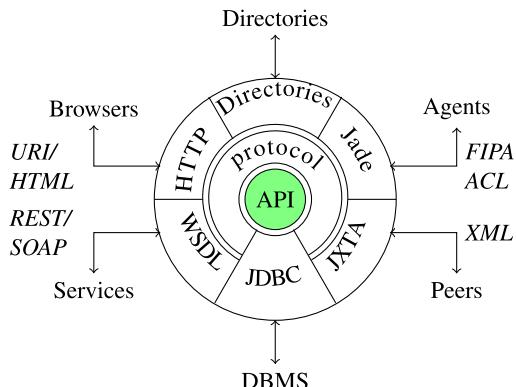
12.7.2 Alignment Server (INRIA)

The Alignment server,³ associated with the Alignment API (Sect. 10.3.6), has been designed as a middleware server available for applications and web browsing. It exposes most of the functions of the API. Alignments and their metadata persist in a database and the server can be accessed from other tools and applications through a versatile interface.

²<http://bioportal.bioontology.org/>.

³<http://alignapi.gforge.inria.fr>.

Fig. 12.4 The Alignment server is built on the Alignment API that is placed on top of a database management system for sharing alignments and is wrapped around a simple protocol. Each access method is a plug-in that interacts with the server through the protocol. Currently, HTML, agent and web service plug-ins are available.



The Alignment server architecture is made of five layers (see Fig. 12.4):

The Alignment API which provides operations for manipulating alignments (Sect. 10.3.6). This means that most of the Alignment API operations (matching, rendering, trimming, etc.) are available from the server and that any class implementing the API (matcher, renderer, evaluator) can extend the server.

A storage system which offers persistent storage and retrieval of alignments. It implements only basic storage and run-time memory caching functions. The storage is made through a DBMS interface and can be replaced by any database management system as long as it is supported by JDBC.

A protocol manager which handles the server protocol. It accepts queries from plug-in interfaces and uses the server resources for answering them.

Protocol drivers which accept incoming queries in a particular communication system and invoke the protocol manager in order to answer them. Current wrappers exist for HTTP, REST, SOAP, and FIPA ACL.

Directory drivers which allow for declaring the server and its content to service directories. Currently, only Oyster support is available.

The Alignment server supports matching ontologies, manipulating, evaluating, storing and sharing alignments as well as generating processors. It does not support alignment edition.

12.7.3 CATCH (Vrije Universiteit Amsterdam)

A consortium of cultural heritage projects has investigated sharing thesauri and alignments between thesauri under the CATCH vocabulary and alignment repository.⁴ In contrast to BioPortal, thesauri considered by CATCH are expressed in

⁴<http://semanticweb.cs.vu.nl/amalgame/>.

SKOS and dedicated to cultural heritage (Sect. 1.2). Instead of using SKOS mappings, alignments are expressed in a variant of the Alignment format (Sect. 10.1.5), in particular because this allows for embedding metadata about correspondences (van der Meij et al. 2010). Access to the server is provided through SOAP and a client. It allows for loading thesauri and alignments. Thesauri cannot be edited. Alignments can be imported and matching methods can be plugged into the system. Both thesauri and alignments are widely searchable by their content and metadata. The persistent repository of thesauri and alignments uses the Sesame RDF database.⁵ It may use several repositories at once, making CATCH a server with a distributed storage. It has been reported that the server hosted 15 alignments amounting to nearly a million correspondences.

12.8 Alignment Evolution

Besides properly processing and storing, alignments have to be managed in order to be ready when it is time for them to be used (Euzenat et al. 2008a). Managing alignments requires keeping them available in servers and making them evolve if necessary. In the alignment life cycle of Fig. 3.1 (p. 57), evolution is featured through dotted arrows. Usually, alignment evolution corresponds to the creation of a new alignment, derived from an existing one. This evolution should be recorded within the alignment metadata (Sect. 10.2) in addition to changes in the structure. An alignment may also evolve due to being no longer useful, being superceded by another one, or more generally, by the addition of further qualification to an alignment. Alignment evolution may also be triggered either by adding or by discarding correspondences manually produced, or by better methods, since new information is available. This is typically the case when more data populating an ontology helps find new correspondences. In such cases, the application of a matching method with the initial alignment as input should evolve the initial alignment.

There are other systematic cases of alignment evolution. As soon as ontologies evolve, new alignments have to be produced following the evolution of the ontology. This can be achieved by transforming the changes made to ontologies into an alignment (from one ontology version to the next one), which can be composed with the old alignment to obtain an updated alignment (see Fig. 12.5). As demonstrated by this example, alignment management can rely on composition of alignments.

12.8.1 ToMAS (*University of Toronto and IBM Almaden*)

ToMAS (Toronto Mapping Adaptation System) is a system that automatically detects and adapts mappings that have become invalid or inconsistent when schemas

⁵<http://www.openrdf.org>.

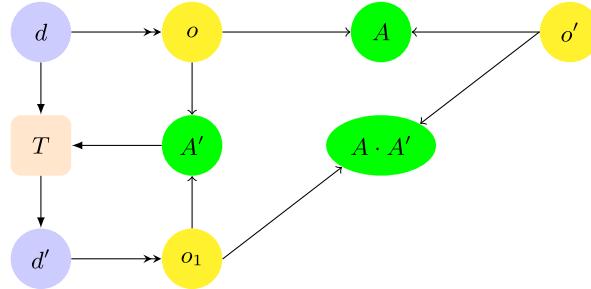


Fig. 12.5 Evolution of alignments (from (Euzenat et al. 2008a)). When an ontology o evolves into a new version o_1 , it is necessary to update the instances of this ontology (d) and the alignment(s) (A) it has with other ontologies (o'). To this extent, a new alignment (A') between the two versions can be established and used for generating the necessary instance transformation (T) and for linking the ontologies o_1 and o' with a new alignment ($A \cdot A'$).

evolve Velegrakis et al. (2004a, 2004b). It is assumed that (i) matching has already been performed, and (ii) correspondences have already been made operational, e.g., as mediators. Since in open environments, such as the web, schemas can evolve without prior notice, some correspondences may become invalid. ToMAS aims at handling such cases, thereby preserving mapping consistency. In particular, it detects mappings affected by structural or constraint changes and it generates automatically the necessary rewritings that are consistent with the updates that have occurred. ToMAS handles relational and XML schemas. It takes two schemas and a set of mappings between them as input. The system works in two phases. First, mappings are analysed and turned into logically valid mappings. During the second step, the result of the previous step is maintained through schema changes. In particular, mappings are modified one by one independently, as appropriate for each kind of change that may occur to the schemas. Three classes of (primitive) schema changes are addressed: (i) operations that change the schema by adding or removing constraints, (ii) modifications to the schema structure by adding or removing elements, and (iii) modifications that reshape schema structure by moving, copying, or renaming elements. The final result of ToMAS is a set of *adapted* mappings which are consistent with the structure and semantics of the evolved schemas.

12.9 Summary

This chapter considered the issue of alignment processing in general. Alignments may be used in different ways for merging, transforming, translating, linking, mediating, and reasoning. Each time, the application may have a fully different way to process alignments.

Some matching systems process directly their results in one of these operations. They do not go through the generation of an alignment. Unfortunately, they cannot

be used for something else. Other systems generate an alignment in a declarative format (Sect. 10.1). These can be exploited either by rendering the alignment into another formation that can be exploited (as SWRL, OWL, C-OWL, Silk scripts, SPARQL CONSTRUCT, SQL, etc.) or by being interpreted by processors, such as mediators.

Useful alignments are such a scarce resource that storing them in an independent format such as those presented in Chap. 10 is very important. This allows for sharing and processing them in different ways independently from the applications. This gives more freedom to application developers to choose the best suited algorithm and to process alignments adequately. In turn, some environments are designed to share and maintain alignments during their whole lifetime. They do not process them but can update them when their exploitation environment requires this.

Part V

Conclusions

Chapter 13

Conclusions

In this book, we have attempted at covering ontology matching in its diversity. In particular, we have shown that many applications may need ontology matching (Chap. 1) and that there are different forms of ontologies that may need to be matched (Chap. 2). Based on reasonable methodological rules (Chap. 3), ontology matching can take advantage of innumerable basic (Chap. 5) and advanced (Chap. 6) techniques composed and supervised in diverse ways (Chap. 7). This has led to a profusion of available systems (Chap. 8). The output of matching can be provided according to different representations (Chap. 10) or executable forms (Chap. 12) which may need to be communicated to users (Chap. 11).

We have provided a systematic view over the resources for helping users, researchers and developers in selecting the system or technique most adapted to their needs. This has been substantiated by identifying application needs (Chap. 1), by classifying matching techniques (Chap. 4) and by proposing adapted methodologies for approaching ontology matching problems (Chap. 3) and by evaluating matching solutions (Chap. 9). This does not mean that, for any application need, the ideal ontology matching solution is directly provided to readers. Techniques presented in this book can be composed in so many ways that the solution space is open-ended and is far from having thoroughly been explored. A lot of research remains to be carried out to develop better solutions, but finding appropriate settings for applications will still require work from users.

In the remainder of this chapter we overview some general trends in the ontology matching field (Sect. 13.1). We present some promising research directions which we believe worth and need further investigations (Sect. 13.2). These stem from all parts of the book. We conclude with general remarks (Sect. 13.3).

13.1 A Brief Outlook of the Trends in the Field

In the past, the ontology matching problem has been addressed in several areas. However, most often this happened in an isolated manner among: (i) database

schemas in the world of information integration, (ii) XML schemas and catalogues on the web, (iii) ontologies (axiomatised theories) in artificial intelligence, semantic web, knowledge representation, and (iv) objects and entities in data mining. Technical issues these areas had encountered were rarely addressed from multidisciplinary and cross-community viewpoints.

During the last decade the areas mentioned above have made substantial progress in matching. However, they require other technologies and cross-fertilisation to continue their growth. This was one of the motivations behind this book and such an initiative as the Ontology Matching web site,¹ which aims at increasing awareness of the existing matching efforts across the relevant communities and at facilitating the cross-fertilisation between them.

Figure 13.1 shows (approximately) how many papers devoted to diverse aspects of matching have been published at various conferences all over the world in recent years.²

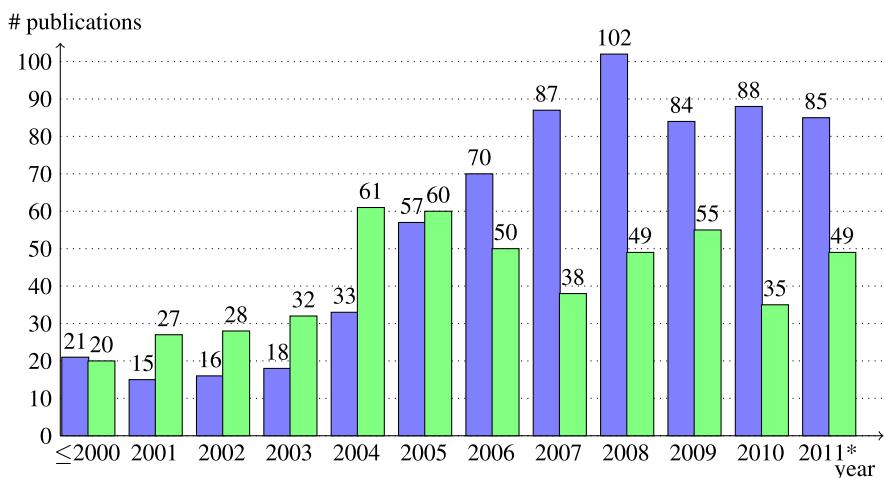


Fig. 13.1 Dynamics of publications devoted to ontology matching (in dark blue: the number of papers from the Publications section of ontologymatching.org; in light green: the number of papers cited in this book). * Value for 2011 is an estimation, since a complete information concerning a year usually becomes stable only when several subsequent years pass (Color figure online).

In the future, we expect a stable level of work on matching (as blue bars of Fig. 13.1 suggest) due to the constant interest in solutions for the semantic heterogeneity problem from both academia and industry. In turn, the green bars suggest that we have considered in this book about 40 (the most salient in our view) papers per year, since the year 2000.

¹<http://www.OntologyMatching.org>.

²Source: www.OntologyMatching.org, Publications section. Accessed: 28.03.2013.

13.2 Future Challenges

The first edition provided a selection of research challenges for ontology matching based on the book organisation in chapters. Since then, we provided an in-depth treatment of these challenges and their developments in (Shvaiko and Euzenat 2008, 2013).

If the design of matchers consists of only tuning further similarity measures or issuing other combinations of matchers, progress may be incremental but no revolution can be expected. Breakthroughs can come from either completely different settings or classes of systems particularly adapted to specific applications. We can expect such improvements from discovering background knowledge, for example, from linked data, as it represents a large and continuously growing source of knowledge. Another source of quality gains is expected from the working environment in which matching is performed. Hence, involving users in matching or social and collaborative matching may provide surprising results.

We briefly give an overview of such challenges in the rest of this section, including

- large-scale and efficient matching (Sect. 13.2.1),
- matching with background knowledge (Sect. 13.2.2),
- matcher selection, combination and tuning (Sect. 13.2.3),
- user involvement (Sect. 13.2.4),
- social and collaborative matching (Sect. 13.2.5),
- uncertainty in ontology matching (Sect. 13.2.6),
- reasoning with alignments (Sect. 13.2.7),
- alignment management: infrastructure and support (Sect. 13.2.8).

Other open issues are the computation of expressive alignments, e.g., correspondences across classes and properties (Sect. 10.1.6), or cross-lingual matching (Sect. 5.2.2). We consider these issues as too specific with respect to the other challenges discussed.

13.2.1 Large-Scale and Efficient Matching

In (Shvaiko and Euzenat 2013), we focussed on large-scale matching evaluation, following some leading matching systems, such as LogMap (Sect. 8.3.26) or GOMMA (Sect. 10.3.3), which are able to scale up. However, this is still not a general trend yet. Thus, scaling and speeding up matching systems remains an issue (Sect. 7.1.1).

Besides quality, the efficiency of matchers is of prime importance in dynamic applications, especially when users cannot wait long for the system to respond or when memory is limited. Current ontology matchers are mostly design-time tools, which are usually not optimised for limited resource consumption. The *execution time* indicates efficiency properties of matchers. However, good execution time can be achieved by using a large amount of *main memory* or bandwidth, compensating the other computational resources, such as CPU. Thus, usage of main memory

should also be measured or improved. Moreover, we expect the need for matching on handheld computers or smartphones in the near future. Overall, the challenge is to come up with scalable ontology matching reference solutions.

13.2.2 Matching with Background Knowledge

One source of difficulty for matching is that ontologies are designed in a particular context, with some background knowledge, which often does not become part of the final ontology specification. Matching can be performed by discovering a common context or background knowledge for the ontologies and using it to extract relations between ontology entities. This context can take different forms, such as a set of resources (web pages, pictures, etc.), that have been annotated with the concepts from an ontology, which provides common anchors to the ontologies to be matched. There remains a lot of experimentation to be conducted to better explore and understand the settings discussed in Sect. 7.3. The difficulty is a matter of balance: adding context, e.g., through linked data, provides new information, and hence, helps in increasing recall, but this new information may also generate incorrect matches, hence decreasing precision.

13.2.3 Matcher Selection, Combination and Tuning

Many matchers are now available. As OAEI campaigns indicate (Chap.9), there is no single matcher that clearly dominates others. Often these perform well in some cases and not so well in some other cases. For both design- and run-time matching, it is necessary to be able to take advantage of the best configuration of matchers. There is evidence from OAEI that matchers do not necessarily find the same correct correspondences. Usually several competing matchers are applied to the same pair of entities in order to increase evidence towards a potential match or mismatch. This mandates solving several important problems: (i) selecting matchers and combining them (Sect. 7.2, Sect. 7.4), and (ii) self-configuring or tuning matchers (Sect. 7.6). On top of this, for dynamic applications it is necessary to perform matcher combination and self-tuning at run time, and thus, the efficiency of the configuration search strategies becomes critical. As the number of available matchers increases, the problem of their selection will become more demanding, e.g., when the task will be to handle more than 50 matchers within one system. Hence, one of the difficulties is of balancing quality and speed of the devised solutions.

13.2.4 User Involvement

In traditional applications, the result of matching performed at design time is screened by human users before being accepted. However, the overwhelming size of

data may render this task difficult. In dynamic applications, users are generally not ontology matching specialists who can be asked to inspect the alignments. Hence, in both cases, user involvement becomes crucial. The challenge is to design ways of involving users so that they can help the matching process without being lost in the amount of results or design choices. At design time, interaction should be both natural and complete; at run time, it should be hidden in user tasks.

In order for users to provide feedback to systems, they need to understand alignments (Sect. 11.3). It is often not sufficient that a matcher returns an alignment, for users to understand it immediately. In order for matching systems to gain a wider acceptance and to be trusted by users, it will be necessary that they provide explanations, at different levels of details, of their results to users or to other programs that exploit them. Generally, the key issue is to provide explanations to users in a simple and clear way in order to facilitate informed decision making.

13.2.5 Social and Collaborative Matching

In an open environment like the web, social support has been key in solving hard and large problems. This approach can also be applied to ontology matching, and therefore, matching could be improved through social interaction (Sect. 11.2). This may be achieved with people explicitly arguing about correspondences or by implicitly voting each time a correspondence is used during an interaction. This calls for algorithms able to rank a large amount of correspondences. The incompleteness and inconsistency of alignments will have to be dealt with in a satisfactory way. Other issues include understanding what tasks are relatively easy for humans, but difficult for machines, how to individuate and deal with malicious users, and which incentive schemes promise to facilitate user participation in establishing alignments collaboratively.

13.2.6 Uncertainty in Ontology Matching

Ontology matching has to improve in dealing with imperfect information. Better understanding the foundations of modelling uncertainty in ontology matching is needed in order to improve detection of correspondences causing inconsistencies or for better qualifying results obtained by exploiting alignments. In dynamic applications, it often occurs that there is no precise correspondence or that an identified correspondence is not specific enough. Hence, it is necessary to choose a good enough one (with respect to application needs). For that purpose, alignments and correspondences must be qualified with respect to their trustworthiness and provenance (Sect. 10.2). In turn, this requires better formalising the link between ontology matching tools and information systems that support this information.

13.2.7 Reasoning with Alignments

The ultimate goal of matching ontologies is to use alignments. For this purpose, alignments should be assigned a semantics and reasoning mechanisms should be developed for this semantics. This may be used in two contexts: (i) when matching, by using the semantics of ontologies and alignments for improving the result of matching, and (ii) after matching, when exploiting the alignments.

The first context requires detecting if the alignment under construction is inconsistent and/or incomplete and restoring consistency and/or completeness if necessary (Sect. 6.5 and Sect. 7.8.2). After matching, reasoning is necessary to know the consequences on one ontology based on its relations with other ontologies. For both purposes, reasoners are necessary to detect inconsistency and consequences. In that respect, we need reasoners able to deal with aligned ontologies and networks of ontologies. The development of such reasoners will rely on the specific semantics of these networks. Their implementations themselves may require that they work in a distributed fashion, e.g., when data cannot be moved.

13.2.8 Alignment Management: Infrastructure and Support

Storing and sharing alignments, as well as collaborative matching, should be supported by adequate tools and infrastructure, especially in dynamic applications. The challenge is to provide convenient and interoperable support, on which tools and applications can rely in order to store and share alignments. This involves using standard ways to communicate alignments and retrieve them. Hence, alignment metadata and annotations should be properly taken into account (Sect. 10.2). The challenge is to provide an alignment support infrastructure at the web scale, thereby stimulating the rise of a web of alignments.

The constant evolution of data, through either dynamic data sources such as semantic sensor networks or linked open data sources commands the parallel evolution of alignments. This requires an infrastructure able to react to the evolution of data and the respective models on a continuous basis. Reacting means adapting alignments and correspondence confidences, automatic matching and debugging of alignments (Sect. 12.8). Evolution of alignments will also trigger further evolution of data and ontologies in a continuously interacting system.

13.3 Final Words

Most often we need to integrate together data sources that were not aiming at their integration while being designed, thus, increasing the difficulty of the matching operation. Even if a good progress has been made in the matching field, as such, ontology matching may appear to be virtually impossible. Indeed, for finding the correspondences between concepts, it is necessary to understand their meaning. Besides

the general meaning ascribed by model-theoretic semantics, the ultimate meaning of concepts is in the head of the people who developed those concepts and we cannot program a computer to learn it.

However, the same remark leads to the conclusion that communication, even between people, is impossible. We know that human beings achieve communication; they at least, succeed quite often in communicating and sometimes fail. Achieving this communication can be viewed as a continuous task of negotiating the relations between concepts, i.e., arguing about alignments, building new ones, questioning them, etc. Therefore, matching ontologies is an on-going work and further substantial progress in the field can be made by considering it in its dynamics.

Appendix A Legends of Figures

We present below the three sets of notations that are used in the pictures of this book.

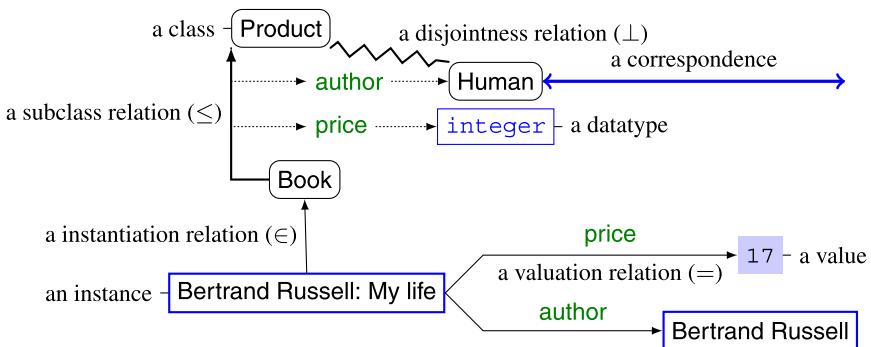


Fig. A.1 Graphic representation of ontologies.

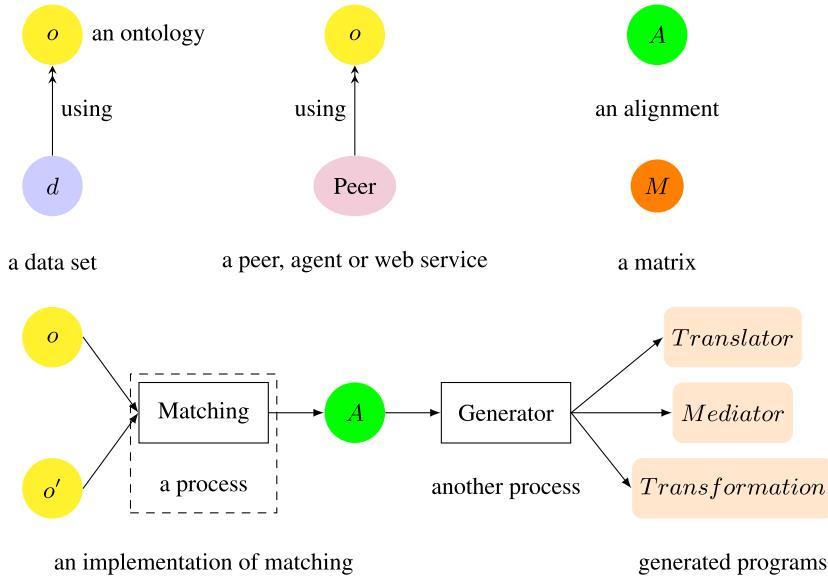


Fig. A.2 Graphic representation of matching application building blocks.

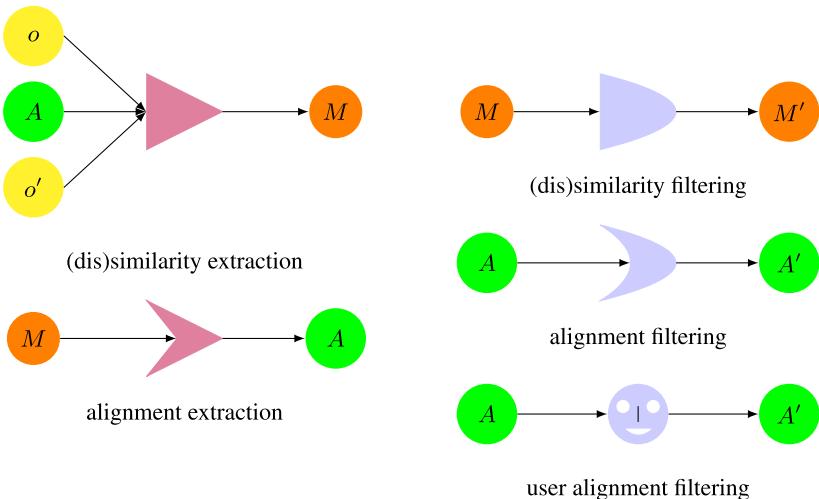


Fig. A.3 Graphic representation of matching system building blocks.

Appendix B

Running Example

The following two ontologies correspond to those displayed in Fig. 2.7: `culture-shop.owl` is the left-hand side ontology and `library.owl` is the right-hand side one.

B.1 culture-shop.owl

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd      "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdf     "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY rdfs   "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY dc      "http://purl.org/dc/elements/1.1/" >
    <!ENTITY owl    "http://www.w3.org/2002/07/owl#" >
    <!ENTITY foaf   "http://xmlns.com/foaf/0.1/">
    <!ENTITY ical   "http://www.w3.org/2002/12/cal/ical#">]>

<rdf:RDF
  xmlns="http://book.ontologymatching.org/example/culture-shop.owl#"
  xml:base="http://book.ontologymatching.org/example/culture-shop.owl#"
  xmlns:foaf ="\&foaf;""
  xmlns:ical ="\&ical;""
  xmlns:rdf ="\&rdf;""
  xmlns:xsd ="\&xsd;""
  xmlns:rdfs ="\&rdfs;""
  xmlns:owl ="\&owl;""
  xmlns:dc ="\&dc;">

  <!-- ##### ONTOLOGY ##### -->

  <owl:Ontology rdf:about="">
    <dc:creator>Jérôme Euzenat</dc:creator>
    <dc:contributor>Pavel Shvaiko</dc:contributor>
    <dc:description>Fragments of a cultural product shop ontology
      </dc:description>
    <dc:date>2006/04/12</dc:date>
    <rdfs:label>Culture shop ontology</rdfs:label>
    <rdfs:comment>An example for the Ontology matching book.
    This ontology fragments organises some cultural product
    the way it could be organised for a cultural product
    e-commerce site.</rdfs:comment>
    <owl:versionInfo>
```

```
$Id: culture-shop.owl,v 1.2 2007/03/15 21:05:21 cvs Exp $
</owl:versionInfo>
</owl:Ontology>

<!-- In OWL-DL all items must be declared -->
<owl:DatatypeProperty rdf:about="#dc;creator" />
<owl:DatatypeProperty rdf:about="#dc;contributor" />
<owl:DatatypeProperty rdf:about="#dc;description" />
<owl:DatatypeProperty rdf:about="#dc;date" />

<!-- ##### CLASSES ##### -->

<owl:Class rdf:ID="Product">
  <rdfs:label xml:lang="en">item</rdfs:label>
  <rdfs:label xml:lang="fr">Marchandise</rdfs:label>
  <rdfs:comment xml:lang="en">The goods which are for sale at our site.
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price" />
      <owl:cardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#name" />
      <owl:maxCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#creator" />
      <owl:maxCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#id" />
      <owl:maxCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#topic" />
      <owl:maxCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="DVD">
  <rdfs:subClassOf rdf:resource="#Product" />
  <rdfs:label xml:lang="en">DVD</rdfs:label>
  <rdfs:comment xml:lang="en"></rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="CD">
  <rdfs:subClassOf rdf:resource="#Product" />
  <rdfs:label xml:lang="en">CD</rdfs:label>
  <rdfs:comment xml:lang="en"></rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Book">
```

```

<rdfs:subClassOf rdf:resource="#Product" />
<rdfs:label xml:lang="en">book</rdfs:label>
<rdfs:comment xml:lang="en">A book.</rdfs:comment>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#author" />
    <owl:cardinality
      rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#publisher" />
    <owl:cardinality
      rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Science">
  <rdfs:subClassOf rdf:resource="#Book" />
  <rdfs:label xml:lang="en">science book</rdfs:label>
  <rdfs:comment xml:lang="en"></rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Textbook">
  <rdfs:subClassOf rdf:resource="#Science" />
  <rdfs:label xml:lang="en">science textbook</rdfs:label>
  <rdfs:comment xml:lang="en">Science book for students.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Popular">
  <rdfs:subClassOf rdf:resource="#Science" />
  <rdfs:label xml:lang="en">popular science book</rdfs:label>
  <rdfs:comment xml:lang="en">
    Science book for a wide audience.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Pocket">
  <rdfs:subClassOf rdf:resource="#Book" />
  <rdfs:label xml:lang="en">pocket book</rdfs:label>
  <rdfs:comment xml:lang="en">
    Paperback bound books of small size.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Children">
  <rdfs:subClassOf rdf:resource="#Book" />
  <rdfs:label xml:lang="en">children book</rdfs:label>
  <rdfs:comment xml:lang="en">Books for children.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Person">
  <rdfs:label xml:lang="en">person</rdfs:label>
  <rdfs:comment xml:lang="en">
    @@Developer: should link with FOAF some day</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Publisher">
  <rdfs:label xml:lang="en">publisher</rdfs:label>
  <rdfs:comment xml:lang="en">A book or music publisher.</rdfs:comment>
</owl:Class>

<!-- ##### PROPERTIES ##### -->

<owl:DatatypeProperty rdf:ID="price">
  <rdfs:domain rdf:resource="#Product" />
  <rdfs:range rdf:resource="xsd:integer" />

```

```

<rdfs:label xml:lang="en">price</rdfs:label>
<rdfs:comment xml:lang="en">The list price of a particular item on
our site. Does not include taxes, shipping or rebates.</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="firstname">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">firstname</rdfs:label>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="lastname">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">lastname</rdfs:label>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#Product" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">name</rdfs:label>
  <rdfs:comment xml:lang="en">The name identifying an item for the common
shoppers.</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="topic">
  <rdfs:domain rdf:resource="#Product" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">topic</rdfs:label>
  <rdfs:comment xml:lang="en">Some (artistic or cultural) topic under which
the item could be classified from a customer standpoint.</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="id">
  <rdfs:domain rdf:resource="#Product" />
  <rdfs:range rdf:resource="&xsd:anyURI" />
  <rdfs:label xml:lang="en">id</rdfs:label>
  <rdfs:comment xml:lang="en">The unique identifier of the item in our infor-
mation system. This is typically the isbn number for books, the
doi for electronic documents, etc.</rdfs:comment>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="creator">
  <rdfs:domain rdf:resource="#Product" />
  <rdfs:range rdf:resource="#Person" />
  <rdfs:label xml:lang="en">creator</rdfs:label>
  <rdfs:comment xml:lang="en">The human creator of a product.</rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="author">
  <rdfs:domain rdf:resource="#Book" />
  <rdfs:range rdf:resource="#Person" />
  <rdfs:label xml:lang="en">author</rdfs:label>
  <rdfs:comment xml:lang="en">The author of a book.</rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publisher">
  <rdfs:domain rdf:resource="#Book" />
  <rdfs:range rdf:resource="#Publisher" />
  <rdfs:label xml:lang="en">publisher</rdfs:label>
  <rdfs:comment xml:lang="en">The publisher of a book.</rdfs:comment>
</owl:ObjectProperty>

<!-- ##### INSTANCES ##### -->

<Popular rdf:about="#a674639524">
  <rdfs:label>Bertrand Russell: My life</rdfs:label>

```

```

<author>
  <Person rdf:about="br">
    <firstname>Bertrand</firstname>
    <lastname>Russell</lastname>
  </Person>
</author>
<publisher>
  <Publisher rdf:about="http://www.routledge.co.uk"/>
</publisher>
<name>My life</name>
<id></id>
<price rdf:datatype="xsd:integer">60</price>
</Popular>

<Book rdf:about="#a6746390923">
  <rdf:type rdf:resource="#Pocket"/>
  <rdfs:label>Albert Camus: La chute</rdfs:label>
  <author>
    <Person rdf:about="ac">
      <firstname>Albert</firstname>
      <lastname>Camus</lastname>
    </Person>
  </author>
  <publisher>
    <Publisher rdf:about="http://www.gallimard.fr"/>
  </publisher>
  <name>La chute</name>
  <id>http://dx.doi.org/10.1002/prot.999</id>
  <price rdf:datatype="xsd:integer">9.95</price>
</Book>

</rdf:RDF>

```

B.2 library.owl

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY foaf "http://xmlns.com/foaf/0.1/">
  <!ENTITY ical "http://www.w3.org/2002/12/cal/ical#">]>

<rdf:RDF
  xmlns="http://book.ontologymatching.org/example/library.owl#"
  xml:base="http://book.ontologymatching.org/example/library.owl#"
  xmlns:foaf = "&foaf;""
  xmlns:ical = "&ical;""
  xmlns:rdf = "&rdf;""
  xmlns:xsd = "&xsd;""
  xmlns:rdfs = "&rdfs;""
  xmlns:owl = "&owl;""
  xmlns:dc = "&dc;">

  <!-- ##### ONTOLOGY ##### -->

  <owl:Ontology rdf:about="">
    <dc:creator>Jérôme Euzenat</dc:creator>
    <dc:contributor>Pavel Shvaiko</dc:contributor>
    <dc:description>Fragments of a library ontology</dc:description>
    <dc:date>2006/04/13</dc:date>

```

```

<rdfs:label>Library ontology</rdfs:label>
<rdfs:comment>An example for the Ontology matching book. This ontology
  fragment provide a first classification for books.</rdfs:comment>
<owl:versionInfo>
  $Id: library.owl,v 1.3 2007/03/15 21:05:21 cvs Exp $
</owl:versionInfo>
</owl:Ontology>

<!-- In OWL-DL all items must be declared -->
<owl:DatatypeProperty rdf:about="#dc:creator" />
<owl:DatatypeProperty rdf:about="#dc:contributor" />
<owl:DatatypeProperty rdf:about="#dc:description" />
<owl:DatatypeProperty rdf:about="#dc:date" />

<!-- ##### CLASSES ##### -->

<owl:Class rdf:ID="Volume">
  <rdfs:label xml:lang="en">volume</rdfs:label>
  <rdfs:label xml:lang="fr">Volume</rdfs:label>
  <rdfs:comment xml:lang="en">Books referenced in the library.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#year" />
      <owl:cardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#author" />
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#title" />
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isbn" />
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Essay">
  <rdfs:subClassOf rdf:resource="#Volume" />
  <rdfs:label xml:lang="en">essay</rdfs:label>
  <rdfs:comment xml:lang="en">A book whose main interest reside in the topic
    considered.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#subject" />
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="LiteraryCritic">
  <rdfs:subClassOf rdf:resource="#Essay" />
  <rdfs:label xml:lang="en">litterary critic</rdfs:label>

```

```

<rdfs:comment xml:lang="en">An essay about Literature.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Politics">
  <rdfs:subClassOf rdf:resource="#Essay" />
  <rdfs:label xml:lang="en">political writings</rdfs:label>
  <rdfs:comment xml:lang="en">An essay about politics.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Biography">
  <rdfs:subClassOf rdf:resource="#Essay" />
  <rdfs:label xml:lang="en">biography</rdfs:label>
  <rdfs:comment xml:lang="en">An essay about a person.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Autobiography">
  <rdfs:subClassOf rdf:resource="#Essay" />
  <rdfs:label xml:lang="en">autobiography</rdfs:label>
  <rdfs:comment xml:lang="en">
    A biography whose author is the subject.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#subject" />
      <owl:allValuesFrom rdf:resource="#Human" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Literature">
  <rdfs:subClassOf rdf:resource="#Volume" />
  <rdfs:label xml:lang="en">literature</rdfs:label>
  <rdfs:comment xml:lang="en">A volume whose main interest reside in
    the treatment of the topic.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Novel">
  <rdfs:subClassOf rdf:resource="#Literature" />
  <rdfs:label xml:lang="en">novel</rdfs:label>
  <rdfs:comment xml:lang="en">A narrative text.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Poetry">
  <rdfs:subClassOf rdf:resource="#Literature" />
  <rdfs:label xml:lang="en">poetry</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Human">
  <rdfs:label xml:lang="en">human</rdfs:label>
  <rdfs:comment xml:lang="en">A Human being.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Writer">
  <rdfs:subClassOf rdf:resource="#Human" />
  <rdfs:label xml:lang="en">writer</rdfs:label>
  <rdfs:comment xml:lang="en">Someone who authors books.</rdfs:comment>
</owl:Class>

<!-- ##### PROPERTIES ##### -->

<owl:DatatypeProperty rdf:ID="year">
  <rdfs:domain rdf:resource="#Volume" />
  <rdfs:range rdf:resource="&xsd;integer" />
  <rdfs:label xml:lang="en">year</rdfs:label>
  <rdfs:comment xml:lang="en">The year of first publication of this edition
    of the volume.</rdfs:comment>
</owl:DatatypeProperty>
```

```

<owl:DatatypeProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#Volume" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">title</rdfs:label>
  <rdfs:comment xml:lang="en">The title of a volume.</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="isbn">
  <rdfs:domain rdf:resource="#Volume" />
  <rdfs:range rdf:resource="&xsd;integer" />
  <rdfs:label xml:lang="en">year</rdfs:label>
  <rdfs:comment xml:lang="en">
    The International Standard Book Number of a volume.</rdfs:comment>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="author">
  <rdfs:domain rdf:resource="#Volume" />
  <rdfs:range rdf:resource="#Writer" />
  <rdfs:label xml:lang="en">author</rdfs:label>
  <rdfs:comment xml:lang="en">The author of a volume.</rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="subject">
  <rdfs:domain rdf:resource="#Essay" />
  <rdfs:range rdf:resource="#owl:Thing" />
  <rdfs:label xml:lang="en">subject</rdfs:label>
  <rdfs:comment xml:lang="en">The subject of an essay.</rdfs:comment>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#Human" />
  <rdfs:range rdf:resource="&xsd:string" />
  <rdfs:label xml:lang="en">name</rdfs:label>
</owl:DatatypeProperty>

<!-- ##### INSTANCES ##### -->

<owl:Thing rdf:about="#a674639524">
  <rdf:type rdf:resource="#Autobiography" />
  <rdfs:label>"My life" by Bertrand Russell</rdfs:label>
  <author>
    <Writer rdf:about="#br">
      <name>Bertrand Russell</name>
    </Writer>
  </author>
  <isbn>0415189853</isbn>
  <subject rdf:resource="#br" />
  <year rdf:datatype="&xsd;integer">1969</year>
  <title>My life</title>
</owl:Thing>

<Novel rdf:about="#a6746390923">
  <rdfs:label>"La chute" by Albert Camus</rdfs:label>
  <author>
    <Writer rdf:about="#ac">
      <name>Albert Camus</name>
    </Writer>
  </author>
  <isbn>2070360105</isbn>
  <year rdf:datatype="&xsd;integer">1956</year>
  <title>La chute</title>
</Novel>

</rdf:RDF>

```

B.3 srcalign.rdf

The following alignment in the Alignment format (see Sect. 10.1.5) is that of Fig. 2.9.

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY ont1 "http://book.ontologymatching.org/example/culture-shop.owl" >
  <!ENTITY ont2 "http://book.ontologymatching.org/example/library.owl" >]>

<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
           xmlns:xsd='&xsd;'>

<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <method>Manually generated (Jérôme Euzenat, 2005/04/13)</method>
  <onto1>
    <Ontology rdf:about="&ont1;">
      <location>file:culture-shop.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about="&ont2;">
      <location>file:culture-shop.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto2>
  <map>
    <Cell>
      <entity1 rdf:resource='&ont1;#name' />
      <entity2 rdf:resource='&ont2;#title' />
      <measure rdf:datatype='&xsd;float'>1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='&ont1;#id' />
      <entity2 rdf:resource='&ont2;#isbn' />
      <measure rdf:datatype='&xsd;float'>.9</measure>
      <relation>&gt;</relation>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='&ont1;#author' />
      <entity2 rdf:resource='&ont2;#author' />
      <measure rdf:datatype='&xsd;float'>1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='&ont1;#Person' />
      <entity2 rdf:resource='&ont2;#Human' />
      <measure rdf:datatype='&xsd;float'>1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>

```

```

    </Cell>
</map>
<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Science' />
    <entity2 rdf:resource='&ont2;#Essay' />
    <measure rdf:datatype='&xsd;float'>.8</measure>
    <relation>&lt;</relation>
  </Cell>
</map>
<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Book' />
    <entity2 rdf:resource='&ont2;#Volume' />
    <measure rdf:datatype='&xsd;float'>.8</measure>
    <relation>&lt;</relation>
  </Cell>
</map>
</Alignment>
</rdf:RDF>

```

B.4 Alternative Alignments for Evaluation

Here are the alignments considered in Chap. 9.

B.4.1 *refalign.rdf*

This is the previous alignment involving only correspondences between classes. It is considered as the reference alignment.

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY ont1 "http://book.ontologymatching.org/example/culture-shop.owl" >
  <!ENTITY ont2 "http://book.ontologymatching.org/example/library.owl" >]>

<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
           xmlns:xsd='&xsd;'>

<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <method>Manually generated (Jérôme Euzenat, 2005/04/13)</method>
  <onto1>
    <Ontology rdf:about="&ont1;">
      <location>file:culture-shop.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about="&ont2;">
      <location>file:library.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto2>
</Alignment>

```

```

        </formalism>
    </Ontology>
</onto2>
<map>
    <Cell>
        <entity1 rdf:resource='&ont1;#Person' />
        <entity2 rdf:resource='&ont2;#Human' />
        <measure rdf:datatype='&xsd;float'>1.0</measure>
        <relation>=</relation>
    </Cell>
</map>
<map>
    <Cell>
        <entity1 rdf:resource='&ont1;#Science' />
        <entity2 rdf:resource='&ont2;#Essay' />
        <measure rdf:datatype='&xsd;float'>.8</measure>
        <relation>&lt;</relation>
    </Cell>
</map>
<map>
    <Cell>
        <entity1 rdf:resource='&ont1;#Book' />
        <entity2 rdf:resource='&ont2;#Volume' />
        <measure rdf:datatype='&xsd;float'>.8</measure>
        <relation>&lt;</relation>
    </Cell>
</map>
</Alignment>
</rdf:RDF>

```

B.4.2 *nearmiss.rdf*

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY ont1 "http://book.ontologymatching.org/example/culture-shop.owl" >
    <!ENTITY ont2 "http://book.ontologymatching.org/example/library.owl" >]>
<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns'
           xmlns:xsd='&xsd;'>
<Alignment>
    <xml>yes</xml>
    <level>0</level>
    <type>*</type>
    <method>Manually generated (Jérôme Euzenat, 2005/04/13)</method>
    <onto1>
        <Ontology rdf:about="&ont1;">
            <location>file:culture-shop.owl</location>
            <formalism>
                <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
            </formalism>
        </Ontology>
    </onto1>
    <onto2>
        <Ontology rdf:about="&ont2;"">
            <location>file:library.owl</location>
            <formalism>
                <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
            </formalism>
        </Ontology>
    </onto2>

```

```

<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Product' />
    <entity2 rdf:resources='&ont2;#Volume' />
    <measure rdf:datatype='&xsd;float'>1.0</measure>
    <relation>=</relation>
  </Cell>
</map>
<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Science' />
    <entity2 rdf:resources='&ont2;#Essay' />
    <measure rdf:datatype='&xsd;float'>.8</measure>
    <relation>&lt;</relation>
  </Cell>
</map>
<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Person' />
    <entity2 rdf:resources='&ont2;#Writer' />
    <measure rdf:datatype='&xsd;float'>.8</measure>
    <relation>&lt;</relation>
  </Cell>
</map>
</Alignment>
</rdf:RDF>

```

B.4.3 farone.rdf

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY ont1 "http://book.ontologymatching.org/example/culture-shop.owl" >
  <!ENTITY ont2 "http://book.ontologymatching.org/example/library.owl" >]>

<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
           xmlns:xsd='&xsd;'>

<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <method>Manually generated (Jérôme Euzenat, 2005/04/13)</method>
  <onto1>
    <Ontology rdf:about="&ont1;">
      <location>file:culture-shop.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about="&ont2;">
      <location>file:library.owl</location>
      <formalism>
        <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
      </formalism>
    </Ontology>
  </onto2>
<map>
  <Cell>
    <entity1 rdf:resource='&ont1;#Book' />

```

```

<entity2 rdf:resource='&ont2;#Volume'/>
<measure rdf:datatype='&xsd;float'>1.0</measure>
<relation>=</relation>
</Cell>
</map>
<map>
<Cell>
<entity1 rdf:resources='&ont1;#Pocket' />
<entity2 rdf:resource='&ont2;#Essay' />
<measure rdf:datatype='&xsd;float'>.8</measure>
<relation>&lt;;</relation>
</Cell>
</map>
<map>
<Cell>
<entity1 rdf:resources='&ont1;#Children' />
<entity2 rdf:resource='&ont2;#Literature' />
<measure rdf:datatype='&xsd;float'>.8</measure>
<relation>&lt;;</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>

```

B.4.4 noncomplete.rdf

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY ont1 "http://book.ontologymatching.org/example/culture-shop.owl" >
    <!ENTITY ont2 "http://book.ontologymatching.org/example/library.owl" >]>

<rdf:RDF xmlns='http://knowledgegeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:base='http://knowledgegeweb.semanticweb.org/heterogeneity/alignment'
           xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns'
           xmlns:xsd='&xsd;'>
<Alignment>
    <xml>yes</xml>
    <level>0</level>
    <type>**</type>
    <method>Manually generated (Jérôme Euzenat, 2013/02/07)</method>
    <onto1>
        <Ontology rdf:about="&ont1;">
            <location>file:culture-shop.owl</location>
            <formalism>
                <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
            </formalism>
        </Ontology>
    </onto1>
    <onto2>
        <Ontology rdf:about="&ont2;">
            <location>file:library.owl</location>
            <formalism>
                <Formalism name="OWL1.0" uri="http://www.w3.org/2002/07/owl#" />
            </formalism>
        </Ontology>
    </onto2>
    <map>
        <Cell>
            <entity1 rdf:resource='&ont1;#Person' />
            <entity2 rdf:resource='&ont2;#Human' />
            <relation>&lt;;</relation>
            <measure rdf:datatype='&xsd;float'>0.7</measure>
        </Cell>
    </map>

```

```
</Cell>
</map>
<map>
<Cell>
  <entity1 rdf:resource='&ont1;#Person' />
  <entity2 rdf:resources='&ont2;#Writer' />
  <relation>></relation>
  <measure rdf:datatype='&xsd;float'>0.6</measure>
</Cell>
</map>
<map>
<Cell>
  <entity1 rdf:resource='&ont1;#Book' />
  <entity2 rdf:resources='&ont2;#Volume' />
  <relation>&lt;</relation>
  <measure rdf:datatype='&xsd;float'>0.7</measure>
</Cell>
</map>
</Alignment>
```

Appendix C

Exercises

The following exercises cover only the technical sections of this book. They are provided to help readers check their understanding of the presented concepts rather than to make assignments to students. Notice that solutions do not represent operational systems; for example, the methodology should be applied to larger data sets. They consist of applications of the presented concepts to a pair of ontologies; they are expected not to be difficult. Due to lack of space, these exercises are applied to small size ontologies. However, interested readers may use their own (larger) ontologies instead. It is certainly worthwhile to use available tools for completing these exercises.

C.1 Applications

C.1 (Application definition) Consider two university data sources dealing with people. The first one is a database developed from the whole university management standpoint, while the second one represents the standpoint of a particular research laboratory. These data sources are managed by different departments and will continue to evolve independently; however, users would like to access them through a unified interface. Obviously, this could be useful, e.g., for checking both lecture and room availability from a single interface.

1. Provide an architecture for this application. For instance, by drawing diagrams similar to those of Chap. 1.
2. What are the requirements to ontology matching in this application with regard to Table 3.1?

C.2 The Matching Problem

C.2 (Ontology representation) Let o be a first data source to be integrated. o is described in English as follows:

- People are divided among Students, Faculty and Staff. Faculty is further divided depending on departments and subdepartments, e.g., Philosophy, Science.
 - People are characterised by their firstname and lastname which are strings, id which is a uri and birthdate which is a date.
 - Students attend Courses which are taught by Faculty and Faculty people have an Office as a room.
 - Pr. Carla Cipolla is a visiting professor in Computer Science and Stefano Zucchini is a PhD Student.
1. Provide a description for o as a folksonomy.
 2. Provide a description for o as a directory.
 3. Provide a description for o as an XML schema.
 4. Provide a description for o as a relational database schema.
 5. Provide a description for o as an entity–relationship schema (or UML diagram).
 6. Provide a description for o as an ontology.

C.3 (Ontology semantics) Assume that the left-hand side ontology of Fig. C.1 is denoted by o .

1. Express o in OWL.
2. Express o as a set of assertions, e.g., $\text{Staff} \sqsubseteq \text{People}$.
3. Provide its semantics.
4. Does $o \models \text{Philosophy} \sqsubseteq \text{People}$ and why?
5. Does $o \models \text{teaches} \sqsubseteq \text{attends}$ and why?

C.3 Classification

C.4 (Kinds of techniques) Consider the structure of ontologies o and o' of Problem C.2, as illustrated in Fig. C.1. Describe which techniques can be used for matching them (see Fig. 4.1) and explain the choices made.

C.4 Basic Techniques

C.5 (Name-based distance computation) Given the ontologies o and o' of Problem C.1 as illustrated in Fig. C.1. Provide the tables (similar to those of Example 5.14 or Example 5.26) for class matching with the following techniques:

1. String distances between all the labels occurring in Fig. C.1:
 - substring similarity,
 - 3-gram similarity,
 - edit distance,
 - Jaro–Winckler measure.

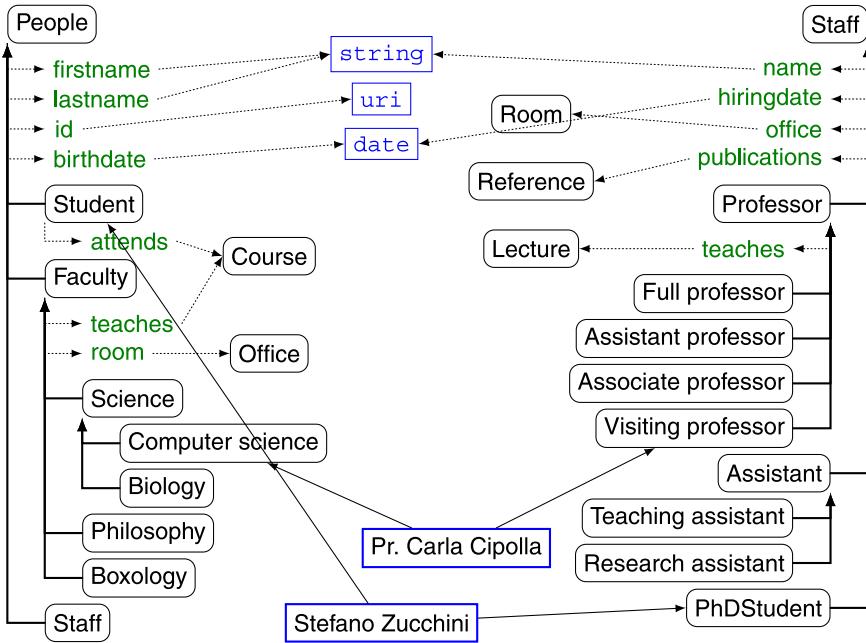


Fig. C.1 Two ontologies to be matched.

2. Linguistic distances between all the labels by using the last version of WordNet:

- cosynonymy similarity,
- gloss overlap,
- Wu–Palmer similarity.

C.6 (Extensional distance computation) Consider the two ontologies o and o' of Problem C.1 as illustrated in Fig. C.1. Assume that the two following tables specify data instances for o and o' , respectively.

Class	firstname	lastname	id	birthdate
Computer Science	G.	Cetriolo	445	04/07/1978
Biology	P.	Pomodoro	1678	01/08/1972
Computer Science	C.	Cipolla	1998	13/06/1977
Philosophy	P.	Carciofo	128	03/09/1982
Student	C.	Fragola	1664	12/12/1985
Biology	A.	Verdura	88	07/09/1981
Student	S.	Zucchini	1178	16/04/1987
Computer Science	F.	di Guava		23/02/1966
Staff	C.	Melocoton	178	14/01/1962

Class	name	hiring date	office
Assistant professor	Giancarlo Cetriolo	2/9/2004	B45
Associate professor	Paola Pomodoro	23/2/2002	B45
Visiting professor	Carla Cipolla	4/12/2007	C17
Assistant professor	Paolo Carciofo		C18
Full professor	Federico diGuava	7/7/1999	B12
Visiting professor	Pierluiggi Pomodoro	4/12/2007	C18
PhDStudent	Mario Staggioni	2/9/2006	B47
PhDStudent	Stefano Zucchini	17/10/2005	B47
PhDStudent	Domenica Melanzana	15/9/2006	

1. Identify which extensional techniques can be used and why.
2. Design a distance for strings which can compare names with and without abbreviated first name.
3. Use the previous string distance for computing a similarity between instances.
4. Use the substring similarity between lastname and name for identifying instances.
5. Starting with each of the previously computed measures, use the single linkage measure on instances for comparing classes of the two ontologies.

C.5 Strategies

C.7 (Measure aggregation) Consider the distances between ontology entities in o and o' given by (i) edit distance computed on their names (see Problem C.5), and (ii) the distance computed with the single linkage measure applied to the substring distance (see Problem C.6).

1. Compute their aggregation with the $\max(x + y - 1, 0)$ triangular norm.
2. Compute their aggregation with the weighted product, such that 2/3 is the weight for the former distance and 1/3 is the weight for the latter one.
3. Compute their aggregation with the weighted sum such that 2/3 is the weight for the former distance and 1/3 is the weight for the latter one.
4. Compute their aggregation with the ordered weighted average such that 2/3 is the weight for the higher distance and 1/3 is the weight for the lower one.

C.8 (Thresholds) Assume that the similarity between entities of ontologies o and o' of Problem C.2 is expressed by the following similarity table:

	People	Student	Faculty	Science	Philosophy	Biology	Staff	Course	Office
Staff	.56	.65	.33	.64	.12	.11	.63	.22	.13
Professor	.62	.36	.60	.40	.44	.32	.55	.21	.36
Assistant	.40	.44	.58	.62	.46	.33	.43	.32	.22
PhDStudent	.64	.92	.45	.60	.65	.52	.55	.33	.34
Room	.12	.20	.20	.18	.10	.12	.09	.11	.62
Reference	.23	.06	.18	.25	.26	.28	.22	.17	.23
Lecture	.15	.16	.26	.23	.34	.12	.14	.20	.16

Provide the set of correspondences resulting from the application of a .6 threshold, specifically:

1. with a hard threshold of .6,
2. with a delta threshold of .6,
3. with a proportional threshold of .6,
4. with a percentage threshold of .6.

C.9 (Alignment extraction) Consider the two ontologies o and o' of Problem C.1 as illustrated in Fig. C.1. Assume that the similarity between their entities is expressed by the similarity table of Problem C.8.

1. Extract an alignment based on the similarity as a stable marriage;
2. Extract an alignment based on the similarity as a pairwise maximal matching;
3. Extract an alignment based on the similarity as the maximum weight graph matching.

C.10 (Composing matchers) Consider the application described in Problem C.1.

1. Provide the architecture for a matching system suitable to match these ontologies (use the answer identified to Problem C.4 for guiding your choice).
2. Compute the alignment with this architecture.

C.6 Evaluation of Matching Systems

C.11 (Precision and recall computation) Let R , as described in Fig. C.2, be the reference alignment between the two ontologies o and o' of Problem C.1 as illustrated in Fig. C.1. Consider the three alignments (A_1, A_2, A_3) as follows:

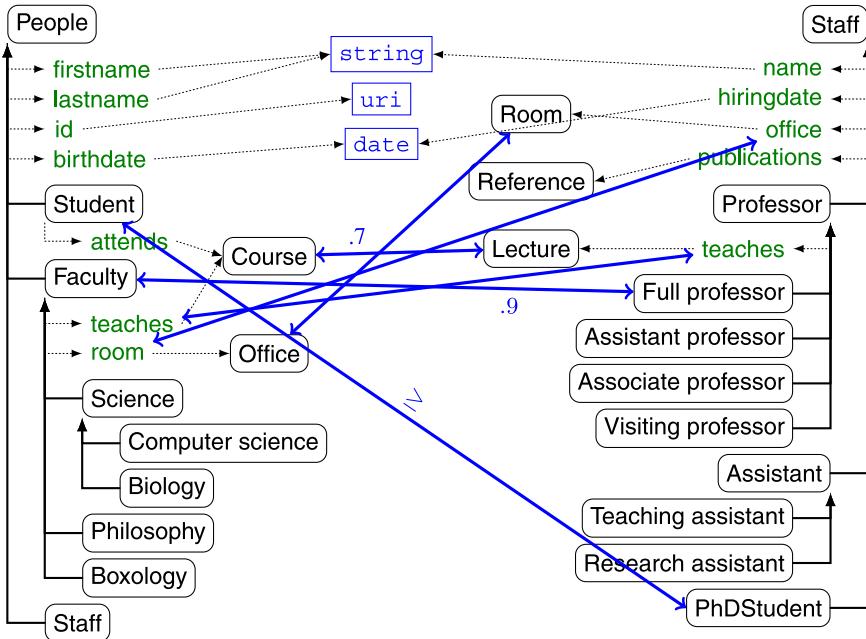


Fig. C.2 Reference alignment R between the ontologies of Fig. C.1. Correspondences are expressed by solid arrows. By default their relation is $=$ and their confidence value is 1.0; otherwise, these are mentioned near the arrows.

room = Room	Office = office	Staff = Staff
firstname \leq name	Student \geq PhDStudent	teaches = teaches (A_1)
Faculty \geq Professor	Office = Room	room = office
People = Staff	Course = Lecture	teaches = teaches (A_2)
Faculty = Full professor	Student = PhDStudent	Staff = Assistant
birthdate = hiringdate	firstname = name	teaching = teaching (A_3)

1. Compute the precision of alignments A_1 , A_2 and A_3 ;
2. Compute the recall of alignments A_1 , A_2 and A_3 ;
3. Compute the F-measure of alignments A_1 , A_2 and A_3 ;
4. Compute the overall of alignments A_1 , A_2 and A_3 ;
5. Compute the Hamming distance between alignments A_1 , A_2 , A_3 and R .

C.12 (Application specific evaluation) Let R , as described in Fig. C.2, be the reference alignment between the two ontologies o and o' of Problem C.1, as illustrated in Fig. C.1. Apply the aggregation technique based on weighted harmonic mean of

Sect. 9.4.1 with the criterion identified for Problem C.1 on alignments A_1 , A_2 and A_3 from Problem C.11.

C.7 Representing Alignments

C.13 (Representation generation) Consider R as described in Fig. C.2.

1. Express it in OWL;
2. Express it in C-OWL;
3. Express it in SWRL;
4. Express it in EDOAL;
5. Express it in SKOS;
6. Discuss the advantages and limitations of these formalisms.

C.8 Explaining Alignments

C.14 (Alignment explanation) Given the ontologies of Fig. C.2 and given the architecture and a solution devised for Problem D.10:

1. Provide a process trace (see for example Fig. 11.9) for a correctly and an incorrectly identified correspondence, namely for a true positive, such as $\text{Student} \geq \text{PhDStudent}$ and for a false positive, such as $\text{Student} = \text{PhDStudent}$;
2. Provide an intuitive explanation, in natural language, for the above correspondences, of why the correspondence under consideration has been correctly or incorrectly identified.

C.9 Processing Alignments

C.15 (Merging ontologies) Let R be the alignment described in Fig. C.2.

1. Describe the merge between o and o' according to R using the OWL import capability;
2. Describe it as one integrated OWL ontology.

C.16 (Data translation) Assume that one wants to transform the data instances from ontology o into instances of o' . Consider the data instances in the first table of Problem C.6.

1. Provide their translation with regard to the reference alignment R described in Fig. C.2.
2. Develop a program able to perform this translation.
3. How data interlinking could be used for improving on the situation? Provide an effective way for doing this and its results.

C.17 (Mediation) Assume that one wants that each of the sources of Problem C.1 can query the other one. It is thus necessary to transform queries and their answers.

1. Given the SPARQL query ‘`SELECT ?room WHERE { ?x rdf:type o:Faculty. ?x o:room ?room .}`’ expressed with respect to ontology o , transform it into a query expressed with regard to ontology o' , according to the reference alignment R described in Fig. C.2.
2. Apply the transformed query to the data instances of the second table of Problem C.6.
3. Develop a mediator able to perform the query transformations and answer translation depending on the reference alignment R described in Fig. C.2.

Appendix D

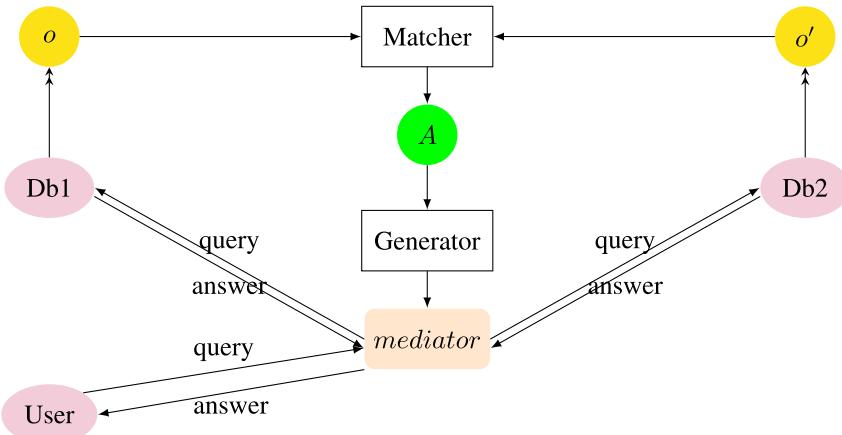
Solutions

We provide sketches of the solutions to the exercises of Appendix C. Most computations are made with the help of Alignment API (version 4.5). The book web site¹ provides scripts and Java classes for using the Alignment API in order to solve some of the questions.

D.1 Applications

Solution D.1 (Application definition) Considering the application presented in Problem C.1,

1. A possible architecture for the application is provided by the following illustration:



2. The requirements of the application with respect to ontology matching could be as follows:

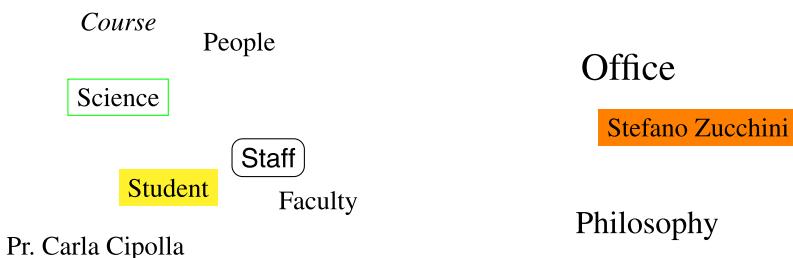
¹<http://book.ontologymatching.org>.

- instances: YES, instances are available.
- run time: NO, matching can be performed at design time. However, since it is said that the database keeps evolving, there is still some alignment maintenance to be carried out.
- automatic: NO, automatic matching is not strictly required and it may be wise to have results checked before putting them in production.
- correct: YES, the alignment must be correct. It would be particularly embarrassing, were it to provide incorrect information.
- complete: YES, the more complete the results, the better. However, having incomplete matching is not critical: this means that some information will be missing in the unified view.
- operation: the operation to be carried out is query mediation, because the databases have to remain independent.

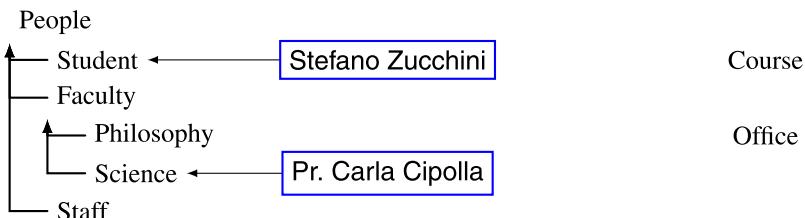
D.2 The Matching Problem

Solution D.2 (Ontology representation) Let o be a first data source to be integrated as defined in Problem C.2.

1. A description for o as a folksonomy could be:



2. Below is a description for o as a directory:



3. Here is a description of o as an XML schema:

```
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.w3.org/2001/XMLSchema">

  <complexType name="People">
    <sequence>
      <element name="firstname" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <element name="lastname" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <element name="birthdate" type="xsd:date" minOccurs="1" maxOccurs="1"/>
      <element name="id" type="xsd:anyURI" minOccurs="1" maxOccurs="1"/>
      <element name="type" type="xsd:string"/>
    </sequence>
  </complexType>

  <complexType name="Student">
    <complexContent>
      <extension base="People">
        <sequence>
          <element name="attends" type="Course" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="Faculty">
    <complexContent>
      <extension base="People">
        <sequence>
          <element name="office" type="Room" maxOccurs="1"/>
          <element name="department" type="Department" minOccurs="1"
                 maxOccurs="1"/>
          <element name="teaches" type="Course" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="Philosophy">
    <complexContent>
      <extension base="Faculty">
        </extension>
    </complexContent>
  </complexType>

  <complexType name="Science">
    <complexContent>
      <extension base="Faculty">
        </extension>
    </complexContent>
  </complexType>

  <complexType name="Course">
    <sequence>
      <element name="title" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <element name="room" type="Room" minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="Departement">
    <sequence>
      <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="Room">
    <sequence>
      <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>
```

```
</complexType>
```

```
</schema>
```

The two instances would be described by the following XML file:

```
<?xml version="1.0" ?>

<Student>
  <firstname>Stefano</firstname>
  <lastname>Zucchini</lastname>
  <birthdate>16/04/1987</birthdate>
  <id>1178</id>
  <type>PhD student</type>
</Student>

<Science>
  <firstname>Carla</firstname>
  <lastname>Cipolla</lastname>
  <birthdate>13/06/1977</birthdate>
  <id>1998</id>
  <type>Visiting professor</type>
  <department>
    <Department><name>Computer science</name></Department>
  </department>
</Science>
</xml>
```

4. An SQL relational database schema corresponding to σ could be:

```
CREATE TABLE People (
  type VARCHAR(20),
  department VARCHAR(40),
  firstname VARCHAR(30),
  lastname VARCHAR(30),
  birthdate DATE,
  id URI,
  office INT,
  PRIMARY KEY (firstname lastname birthdate id)
);

CREATE TABLE Department (
  name VARCHAR(40),
  PRIMARY KEY (name)
);

CREATE TABLE Room (
  number INT,
  PRIMARY KEY (number)
);

CREATE TABLE Course (
  title VARCHAR(100),
  id VARCHAR(10),
  room INT,
  PRIMARY KEY (id)
);

CREATE TABLE teaches (
  course INT,
  firstname VARCHAR(30),
  lastname VARCHAR(30),
  birthdate DATE,
  id URI
);

CREATE TABLE attends (
```

```

course INT,
firstname VARCHAR(30),
lastname VARCHAR(30),
birthdate DATE,
id URI
);

INSERT INTO People (firstname, lastname, birthdate, id, type, department)
VALUES ('Carla', 'Cipolla', '13/06/1977', 1998, 'Visiting professor',
'Computer science');

INSERT INTO People (firstname, lastname, birthdate, id, type)
VALUES ('Stefano', 'Zucchini', '16/04/1987', 1178, 'PhD student');

```

5. Figure D.1 is a description for o as a UML class diagram. Instances do not appear there.

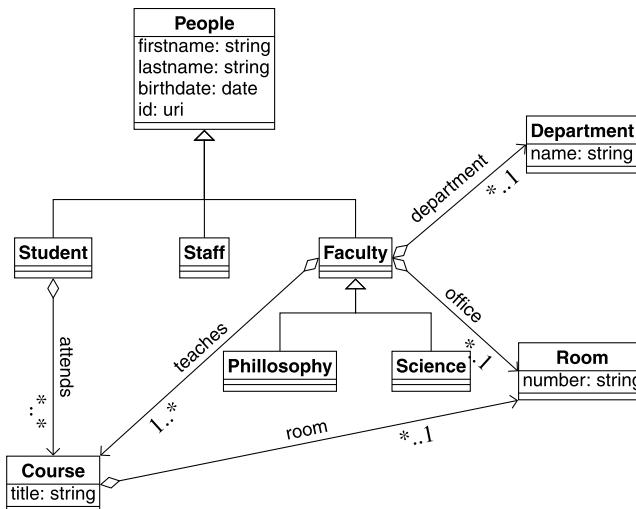


Fig. D.1 Representation of o as a UML class diagram.

6. Figure D.2 provides a description of o as an ontology.

Solution D.3 (Ontology semantics) Assume that the left-hand side ontology of Fig. C.1 is denoted by o .

1. o can be expressed in OWL as follows:

```

<owl:Class rdf:about="#People">
  <rdfs:subClassOf
    <owl:Restriction>
      <owl:onProperty rdf:resource="#firstname"/>
      <owl:allValuesFrom rdf:resources="xsd:string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf
    <owl:Restriction>

```

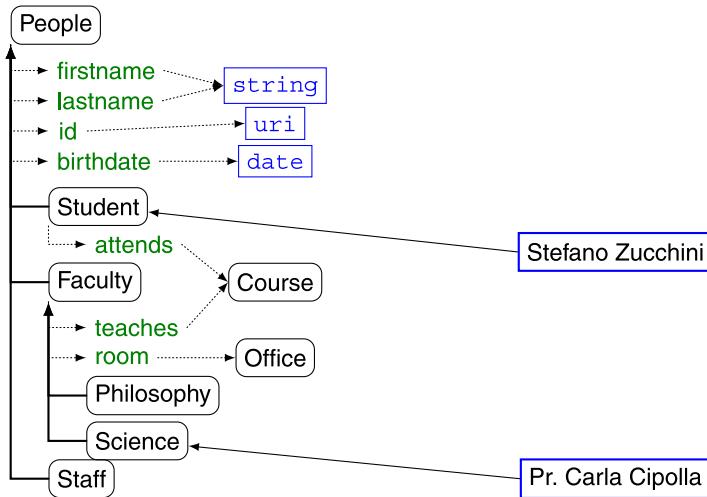


Fig. D.2 Representation of *o* as an ontology.

```

<owl:onProperty rdf:resource="#lastname" />
<owl:allValuesFrom rdf:resources="xsd:string" />
</owl:Restriction>
</rdf:subClassOf>
<rdfs:subClassOf
  <owl:Restriction>
    <owl:onProperty rdf:resource="#id" />
    <owl:allValuesFrom rdf:resources="xsd:uri" />
  </owl:Restriction>
</rdf:subClassOf>
<rdfs:subClassOf
  <owl:Restriction>
    <owl:onProperty rdf:resource="#birthdate" />
    <owl:allValuesFrom rdf:resources="xsd:date" />
  </owl:Restriction>
</rdf:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Student">
  <rdfs:subClassOf rdf:resource="#People" />
  <rdfs:subClassOf
    <owl:Restriction>
      <owl:onProperty rdf:resource="#attends" />
      <owl:allValuesFrom rdf:resources="#Course" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Faculty">
  <rdfs:subClassOf rdf:resource="#People" />
  <rdfs:subClassOf
    <owl:Restriction>
      <owl:onProperty rdf:resource="#teaches" />
      <owl:allValuesFrom rdf:resources="#Course" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf
    <owl:Restriction>
      <owl:onProperty rdf:resource="#room" />
      <owl:allValuesFrom rdf:resources="#Office" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

```

</rdf:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Science">
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>
<owl:Class rdf:about="#Computer science">
  <rdfs:subClassOf rdf:resource="#Science"/>
</owl:Class>
<owl:Class rdf:about="#Biology">
  <rdfs:subClassOf rdf:resource="#Science"/>
</owl:Class>
<owl:Class rdf:about="#Philosophy">
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>
<owl:Class rdf:about="#Boxology">
  <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>
<owl:Class rdf:about="#Staff">
  <rdfs:subClassOf rdf:resource="#People"/>
</owl:Class>
<owl:Student rdf:about="#Stefano Zucchini" />
<owl:Computer Science rdf:about="#Pr. Carla Cipolla" />

```

2. o can be expressed in the simplified description logic format as follows:

$$\begin{array}{ll}
 \text{People} \sqsubseteq \forall \text{firstname.string} & \text{People} \sqsubseteq \forall \text{lastname.string} \\
 \text{People} \sqsubseteq \forall \text{id.uri} & \text{People} \sqsubseteq \forall \text{birthdate.date} \\
 \text{Student} \sqsubseteq \text{People} & \text{Student} \sqsubseteq \forall \text{attends.Course} \\
 \text{Faculty} \sqsubseteq \text{People} & \\
 \text{Faculty} \sqsubseteq \forall \text{teaches.Course} & \text{Faculty} \sqsubseteq \forall \text{room.Office} \\
 \text{Science} \sqsubseteq \text{Faculty} & \text{Computer science} \sqsubseteq \text{Science} \\
 \text{Biology} \sqsubseteq \text{Science} & \text{Philosophy} \sqsubseteq \text{Faculty} \\
 \text{Boxology} \sqsubseteq \text{Faculty} & \text{Staff} \sqsubseteq \text{People} \\
 \text{Stefano Zucchini} \in \text{Student} & \text{Pr. Carla Cipolla} \in \text{Computer science}
 \end{array}$$

3. The models of o are those interpretations m over a domain D such that:

$$\begin{aligned}
 m(\text{teaches}) &\subseteq D \times D \\
 m(\text{attends}) &\subseteq D \times D \\
 m(\text{People}) &\subseteq D \\
 &\cap \{x \in D; \forall y; (x, y) \in m(\text{firstname}), y \in \mathbb{S}\} \\
 &\cap \{x \in D; \forall y; (x, y) \in m(\text{lastname}), y \in \mathbb{S}\} \\
 &\cap \{x \in D; \forall y; (x, y) \in m(\text{id}), y \in \mathbb{U}\} \\
 &\cap \{x \in D; \forall y; (x, y) \in m(\text{birthdate}), y \in \mathbb{D}\} \\
 m(\text{Student}) &\subseteq m(\text{People})
 \end{aligned}$$

$$\begin{aligned}
& \cap \{x \in D; \forall y; \langle x, y \rangle \in m(\text{attends}), y \in m(\text{Course})\} \\
m(\text{Faculty}) & \subseteq m(\text{People}) \\
& \cap \{x \in D; \forall y; \langle x, y \rangle \in m(\text{teaches}), y \in m(\text{Course})\} \\
& \cap \{x \in D; \forall y; \langle x, y \rangle \in m(\text{room}), y \in m(\text{Office})\} \\
m(\text{Science}) & \subseteq m(\text{Faculty}) \\
m(\text{Computer science}) & \subseteq m(\text{Science}) \\
m(\text{Biology}) & \subseteq m(\text{Science}) \\
m(\text{Philosophy}) & \subseteq m(\text{Faculty}) \\
m(\text{Boxology}) & \subseteq m(\text{Faculty}) \\
m(\text{Staff}) & \subseteq m(\text{People}) \\
m(\text{Stefano Zucchini}) & \in m(\text{Student}) \\
m(\text{Pr. Carla Cipolla}) & \in m(\text{Computer science})
\end{aligned}$$

4. $o \models \text{Philosophy} \sqsubseteq \text{People}$ because for any model m of o , $m(\text{Philosophy}) \subseteq m(\text{Faculty})$ and $m(\text{Faculty}) \subseteq m(\text{People})$. Hence $m(\text{Philosophy}) \subseteq m(\text{People})$.
5. $o \not\models \text{teaches} \sqsubseteq \text{attends}$ because the interpretation I with one course c , one student s and one faculty f such that $I(\text{People}) = \{s, t\}$, $I(\text{Course}) = \{c\}$, $I(\text{Student}) = \{s\}$, $I(\text{Computer science}) = I(\text{Science}) = I(\text{Faculty}) = \{f\}$, $I(\text{Stefano Zucchini}) = s$, $I(\text{Pr. Carla Cipolla}) = f$, $I(\text{attends}) = \{\langle s, c \rangle\}$, and $I(\text{teaches}) = \{\langle f, c \rangle\}$, all other terms having an empty interpretation, is a model of the ontology and it does not satisfy the assertion.

D.3 Classification

Solution D.4 (Kinds of techniques) We believe that all the techniques considered in our classifications can be used:

- terminological: because the two ontologies are expressed in the same language. However, the poor overlap between the two sets of terms can be the sign of a poor overlap between the ontologies or require lexicon-based methods;
- structural: because there are constraints on properties, but we need anchors;
- extensional: because there are instances; this is invaluable, especially that they can easily be linked;
- semantic: also needs anchors.

D.4 Basic Techniques

Solution D.5 (Name-based distance computation) Given the ontologies o and o' of Problem C.1 as illustrated in Fig. C.1.

1. Below are the distances or similarities, between all the class labels occurring in Fig. C.1, obtained with the OntoSim library (version 2.1).²

- substring distance:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.87	0.88	0.88	0.75	0.57	0.87	1.00	1.00	0.88	0.89	0.92
Reference	0.87	0.50	0.88	0.75	0.86	0.73	1.00	0.87	1.00	1.00	0.67
AssistantProfessor	0.92	0.92	0.92	0.84	0.74	0.83	0.92	0.92	0.92	0.86	0.94
VisitingProfessor	0.91	0.92	0.92	0.92	0.91	0.83	0.92	0.91	0.92	0.85	0.94
ResearchAssistant	0.83	0.92	0.92	0.83	0.73	0.91	1.00	0.91	0.92	0.93	0.94
TeachingAssistant	0.91	0.92	0.83	0.83	0.73	0.91	0.92	0.91	0.92	0.85	0.88
Professor	0.87	0.88	0.88	0.88	0.86	0.73	0.88	0.87	0.88	0.79	0.92
Room	0.80	1.00	1.00	1.00	1.00	0.80	0.83	0.80	0.82	0.86	0.79
PhDStudent	0.88	0.76	0.88	0.18	0.73	0.88	1.00	0.88	1.00	0.80	0.84
Lecture	0.69	0.86	0.86	0.71	0.83	0.85	0.87	0.69	0.86	0.88	0.91
FullProfessor	0.89	0.90	0.80	0.90	0.89	0.79	0.90	0.89	0.90	0.83	0.93
AssociateProfessor	0.92	0.84	0.92	0.92	0.91	0.83	0.92	0.92	0.92	0.86	0.88
Staff	0.82	0.83	0.83	0.67	0.00	0.64	1.00	1.00	1.00	0.87	0.90

- 3-gram distance:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	1.00	1.00	1.00	1.00	0.80	1.00	1.00	1.00	1.00	1.00	1.00
Reference	1.00	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.80
AssistantProfessor	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00	1.00	1.00	1.00
VisitingProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ResearchAssistant	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00	1.00	1.00	1.00
Professor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Room	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PhDStudent	1.00	1.00	1.00	0.23	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Lecture	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssociateProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Staff	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00

²<http://ontosim.gforge.inria.fr>.

There is not much in common between the trigrams of these labels: trigrams are more effective on longer texts.

- edit distance:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.89	0.78	0.89	0.67	0.67	0.89	1.00	1.00	0.89	0.90	0.87
Reference	0.78	0.56	0.89	0.78	1.00	0.67	1.00	0.78	1.00	1.00	0.60
AssistantProfessor	0.89	0.78	0.94	0.83	0.78	0.89	0.94	0.89	0.89	0.89	0.89
VisitingProfessor	0.88	0.82	1.00	0.82	0.82	0.88	0.94	0.88	0.88	0.88	0.94
ResearchAssistant	0.88	0.82	0.82	0.82	0.82	0.94	1.00	0.94	0.94	0.88	0.88
TeachingAssistant	0.88	0.88	0.82	0.82	0.82	0.94	0.94	0.94	0.94	0.82	0.88
Professor	0.78	0.89	1.00	0.89	0.89	0.78	0.89	0.78	0.89	0.80	0.87
Room	0.83	1.00	1.00	1.00	1.00	1.00	0.75	0.83	0.71	0.80	0.93
PhDStudent	0.90	0.80	0.90	0.30	0.80	0.90	1.00	0.80	1.00	0.80	0.80
Lecture	0.57	0.86	0.86	0.86	0.86	0.86	1.00	0.71	1.00	0.90	0.80
FullProfessor	0.85	0.92	0.92	0.92	0.92	0.85	0.92	0.85	0.85	0.85	0.93
AssociateProfessor	0.83	0.72	0.89	0.83	0.83	0.89	0.89	0.83	0.89	0.89	0.78
Staff	1.00	0.86	1.00	0.71	0.00	1.00	1.00	1.00	1.00	0.90	0.93

- Jaro–Winckler measure:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.72	0.75	0.68	0.67	0.78	0.74	1.00	1.00	0.75	0.89	0.83
Reference	0.84	0.70	0.75	0.78	0.80	0.66	1.00	0.64	1.00	1.00	0.74
AssistantProfessor	0.74	0.80	0.71	0.73	0.80	0.76	1.00	1.00	0.76	0.81	0.76
VisitingProfessor	0.75	0.86	0.76	0.81	1.00	0.73	0.76	1.00	0.81	0.75	0.73
ResearchAssistant	0.86	0.84	1.00	0.72	1.00	0.91	1.00	0.71	1.00	0.71	0.75
TeachingAssistant	1.00	0.84	0.72	0.82	0.70	0.91	0.76	0.75	0.76	0.72	0.79
Professor	0.76	0.82	1.00	0.75	0.80	0.70	0.75	0.70	0.70	0.66	0.72
Room	0.67	1.00	1.00	1.00	1.00	0.67	0.65	0.67	0.64	0.98	0.72
PhDStudent	0.77	0.93	0.90	0.68	1.00	0.75	1.00	0.79	1.00	0.68	0.73
Lecture	0.55	0.80	0.76	0.65	1.00	0.68	1.00	0.68	0.97	0.98	0.75
FullProfessor	0.69	0.76	0.65	0.76	1.00	0.80	0.75	0.77	0.80	0.74	0.72
AssociateProfessor	0.90	0.79	0.68	0.81	0.80	0.79	0.85	0.74	0.83	0.79	0.71
Staff	1.00	0.73	0.73	0.66	0.40	0.69	1.00	1.00	1.00	1.00	1.00

- For computing linguistic measures, we have used WordNet 3.0, JWNL 1.4, Lucene 3.0.2 through the OntoSim library and the Alignment API 4.5:

- cosynonymy distance:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reference	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssistantProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
VisitingProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Room	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PhDStudent	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Lecture	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssociateProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Staff	1.00	1.00	0.89	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00

This distance is not very useful because few synonyms occur in the classes.

- gloss overlap similarity (as a distance):

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.98	1.00	1.00	0.93	0.94	0.98	1.00	1.00	1.00	0.97	1.00
Reference	0.96	0.98	1.00	0.97	0.98	0.96	1.00	0.99	0.99	0.97	1.00
AssistantProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
VisitingProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	0.98	1.00	0.94	0.92	0.98	1.00	1.00	0.96	1.00	1.00	1.00
Room	0.94	1.00	0.97	0.94	0.97	1.00	1.00	0.97	0.97	0.97	1.00
PhDStudent	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Lecture	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97	1.00
FullProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssociateProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Staff	0.92	1.00	0.92	0.98	0.00	0.97	1.00	0.97	1.00	0.97	1.00

- Wu-Palmer similarity (as a distance):

	Course	Science	Faculty	Student	Staff	Office	Biology	People	Philosophy	ComputerScience	
Assistant	0.50	1.00	1.00	0.40	0.57	0.42	1.00	1.00	0.50	1.00	1.00
Reference	0.12	0.43	0.50	0.52	0.40	0.36	1.00	0.75	0.43	0.38	1.00
AssistantProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
VisitingProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	0.58	1.00	1.00	0.54	0.65	0.48	1.00	1.00	0.57	1.00	1.00
Room	0.20	0.78	0.25	0.47	0.33	0.22	1.00	0.50	0.55	0.80	1.00
PhDStudent	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Lecture	0.29	0.67	0.67	1.00	0.67	0.29	1.00	0.80	0.62	0.69	1.00
FullProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssociateProfessor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Staff	0.29	0.80	0.00	0.44	0.00	0.09	1.00	0.64	0.52	0.82	1.00

Solution D.6 (Extensional distance computation) Given the ontologies o and o' of Problem C.1 as illustrated in Fig. C.1; we assume that the two tables of Problem C.6 specify data instances for o and o' , respectively.

1. In order to use the extensions of classes for matching o and o' , it is first necessary to produce record linkage because there are no ‘common extension’ nor easy way to identify objects.
2. It is clear that `firstname` and `lastname` in the first table have to be compared with `name` in the second one. An easy approximation would be to check if the `lastname` is a suffix for `name`. This joined with the test that `firstname` is also a prefix (of length 1) for `name` should provide a good guess. Unfortunately, it does not help distinguish between Paola and Pierluiggi Pomodoro. It is also not enough for identifying Federico di Guava since the name has not be written in the same way. Further normalisation of the names should certainly be performed (identifying first and last names in the second table and comparing them to the first one). One good record linkage technique for these tables would be:
 - (a) name_2 is name with the first term (alternatively all terms but the last) acronymed, i.e., replaced by their first letter followed with a dot ‘.’;
 - (b) name_3 is name_2 with all capital letters, but the first in each name, preceded by a space ‘ ’;
 - (c) name'_2 is the concatenation of `firstname`, a space and `lastname`;
 - (d) compute edit distance between name_3 and name'_2 .
3. This distance will provide the following results:

	128	1998	1678	1178	88	445	1664	unknown	178
fdg	0.90	0.70	0.90	0.90	0.70	0.90	0.80	0.00	0.91
gc	0.50	0.70	0.70	0.90	0.70	0.00	0.70	0.90	0.64
sz	0.80	0.90	0.90	0.00	0.90	0.90	0.90	0.90	0.82
plp	0.60	0.80	0.00	0.90	0.70	0.70	0.80	0.90	0.64
cc	0.80	0.00	0.80	0.90	0.78	0.70	0.56	0.70	0.73
pc	0.00	0.80	0.60	0.80	0.80	0.50	0.70	0.90	0.73
ms	0.64	0.91	0.82	0.64	0.91	0.73	0.64	0.91	0.82
pp	0.60	0.80	0.00	0.90	0.70	0.70	0.80	0.90	0.64
dm	0.91	0.82	0.91	0.82	0.73	0.82	0.73	0.73	0.64

4. The substring similarity yields:

	128	1998	1678	1178	88	445	1664	unknown	178
fdg	0.92	0.91	0.92	0.92	0.83	0.83	0.91	0.58	0.84
gc	0.85	0.84	0.92	0.92	0.92	0.38	0.84	0.92	0.85
sz	0.92	0.91	0.92	0.33	0.91	0.92	0.91	0.92	0.92
plp	0.92	0.92	0.38	0.92	0.84	0.92	0.92	0.85	0.93
cc	0.71	0.30	0.90	0.90	0.90	0.81	0.80	0.90	0.91
pc	0.27	0.81	0.91	0.91	0.90	0.73	0.81	0.91	0.83
ms	0.83	0.91	0.91	0.83	0.91	0.74	0.82	0.91	0.83
pp	0.91	0.81	0.27	1.00	0.90	0.82	0.71	0.91	0.91
dm	0.92	0.84	0.85	0.85	0.92	0.92	0.84	0.92	0.78

5. Single linkage applied to the results of the first method comparing individuals (Answer 3) provides:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reference	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssistantProfessor	1.00	0.00	0.00	0.70	0.64	1.00	1.00	0.64	0.60	0.00	0.00
VisitingProfessor	1.00	0.00	0.60	0.56	0.64	1.00	1.00	0.56	0.00	0.60	0.00
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	1.00	0.00	0.00	0.56	0.64	1.00	1.00	0.56	0.00	0.00	0.00
Room	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PhDStudent	1.00	0.73	0.64	0.00	0.64	1.00	1.00	0.00	0.73	0.64	0.73
Lecture	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullProfessor	1.00	0.00	0.90	0.80	0.91	1.00	1.00	0.80	0.70	0.90	0.00
AssociateProfessor	1.00	0.00	0.60	0.80	0.64	1.00	1.00	0.64	0.00	0.60	0.70
Staff	1.00	0.73	0.64	0.00	0.64	1.00	1.00	0.00	0.73	0.64	0.73

and the substring comparison yields:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reference	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssistantProfessor	1.00	0.38	0.27	0.81	0.83	1.00	1.00	0.81	0.90	0.27	0.38
VisitingProfessor	1.00	0.30	0.71	0.80	0.91	1.00	1.00	0.80	0.38	0.71	0.30
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	1.00	0.27	0.27	0.71	0.83	1.00	1.00	0.71	0.27	0.27	0.30
Room	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PhDStudent	1.00	0.74	0.83	0.33	0.78	1.00	1.00	0.33	0.85	0.83	0.74
Lecture	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullProfessor	1.00	0.58	0.92	0.91	0.84	1.00	1.00	0.84	0.83	0.92	0.58
AssociateProfessor	1.00	0.27	0.91	0.71	0.91	1.00	1.00	0.71	0.27	0.91	0.81
Staff	1.00	0.74	0.83	0.33	0.78	1.00	1.00	0.33	0.85	0.83	0.74

D.5 Strategies

Solution D.7 (Measure aggregation) Consider the distances between ontology entities in o and o' given by (i) edit distance computed on their names (see Solution D.5, third distance), and (ii) the distance computed with the single linkage measure applied to the substring distance (see Solution D.6, item 3).

1. The aggregation with the $\max(x + y - 1, 0)$ triangular norm is as follows:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reference	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
AssistantProfessor	1.00	0.78	0.94	0.83	0.83	1.00	1.00	0.89	0.90	0.89	0.89
VisitingProfessor	1.00	0.82	1.00	0.82	0.91	1.00	1.00	0.88	0.88	0.88	0.94
ResearchAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TeachingAssistant	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Professor	1.00	0.89	1.00	0.89	0.89	1.00	1.00	0.78	0.89	0.80	0.87
Room	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PhDStudent	1.00	0.80	0.90	0.33	0.80	1.00	1.00	0.80	1.00	0.83	0.80
Lecture	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullProfessor	1.00	0.92	0.92	0.92	0.92	1.00	1.00	0.85	0.85	0.92	0.93
AssociateProfessor	1.00	0.72	0.91	0.83	0.91	1.00	1.00	0.83	0.89	0.91	0.81
Staff	1.00	0.86	1.00	0.71	0.78	1.00	1.00	1.00	1.00	0.90	0.93

2. The aggregation with the weighted product, such that 2/3 is the weight for the former distance and 1/3 is the weight for the latter one, is:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.20	0.17	0.20	0.15	0.15	0.20	0.22	0.22	0.20	0.20	0.19
Reference	0.17	0.12	0.20	0.17	0.22	0.15	0.22	0.17	0.22	0.22	0.13
AssistantProfessor	0.20	0.07	0.06	0.15	0.14	0.20	0.21	0.16	0.18	0.05	0.08
VisitingProfessor	0.20	0.05	0.16	0.15	0.17	0.20	0.21	0.16	0.08	0.14	0.06
ResearchAssistant	0.20	0.18	0.18	0.18	0.18	0.21	0.22	0.21	0.21	0.20	0.20
TeachingAssistant	0.20	0.20	0.18	0.18	0.18	0.21	0.21	0.21	0.21	0.18	0.20
Professor	0.17	0.05	0.06	0.14	0.16	0.17	0.20	0.12	0.05	0.05	0.06
Room	0.19	0.22	0.22	0.22	0.22	0.22	0.17	0.19	0.16	0.18	0.21
PhDStudent	0.20	0.13	0.17	0.02	0.14	0.20	0.22	0.06	0.19	0.15	0.13
Lecture	0.13	0.19	0.19	0.19	0.19	0.19	0.22	0.16	0.22	0.20	0.18
FullProfessor	0.19	0.12	0.19	0.19	0.17	0.19	0.21	0.16	0.16	0.17	0.12
AssociateProfessor	0.19	0.04	0.18	0.13	0.17	0.20	0.20	0.13	0.05	0.18	0.14
Staff	0.22	0.14	0.18	0.05	0.00	0.22	0.22	0.07	0.19	0.17	0.15

3. The aggregation with the weighted sum, such that 2/3 is the weight for the former distance and 1/3 is the weight for the latter one, is:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.93	0.85	0.93	0.78	0.78	0.93	1.00	1.00	0.93	0.93	0.91
Reference	0.85	0.70	0.93	0.85	1.00	0.78	1.00	0.85	1.00	1.00	0.73
AssistantProfessor	0.93	0.65	0.72	0.83	0.79	0.93	0.96	0.86	0.89	0.68	0.72
VisitingProfessor	0.92	0.65	0.90	0.82	0.85	0.92	0.96	0.85	0.72	0.83	0.73
ResearchAssistant	0.92	0.88	0.88	0.88	0.88	0.96	1.00	0.96	0.96	0.92	0.92
TeachingAssistant	0.92	0.92	0.88	0.88	0.88	0.96	0.96	0.96	0.96	0.88	0.92
Professor	0.85	0.68	0.76	0.83	0.87	0.85	0.93	0.76	0.68	0.62	0.68
Room	0.89	1.00	1.00	1.00	1.00	1.00	0.83	0.89	0.81	0.87	0.96
PhDStudent	0.93	0.78	0.88	0.31	0.79	0.93	1.00	0.64	0.95	0.81	0.78
Lecture	0.71	0.90	0.90	0.90	0.90	0.90	1.00	0.81	1.00	0.93	0.87
FullProfessor	0.90	0.81	0.92	0.92	0.90	0.90	0.95	0.84	0.84	0.87	0.82
AssociateProfessor	0.89	0.57	0.90	0.79	0.86	0.93	0.93	0.79	0.68	0.90	0.79
Staff	1.00	0.82	0.94	0.59	0.26	1.00	1.00	0.78	0.95	0.88	0.87

4. The aggregation with the ordered weighted average, such that 2/3 is the weight for the higher distance and 1/3 is the weight for the lower one, is:

	Course	Science	Faculty	Student	Staff	Office	Boxology	People	Biology	Philosophy	ComputerScience
Assistant	0.96	0.93	0.96	0.89	0.89	0.96	1.00	1.00	0.96	0.97	0.96
Reference	0.93	0.85	0.96	0.93	1.00	0.89	1.00	0.93	1.00	1.00	0.87
AssistantProfessor	0.96	0.65	0.72	0.83	0.81	0.96	0.98	0.86	0.90	0.68	0.72
VisitingProfessor	0.96	0.65	0.90	0.82	0.88	0.96	0.98	0.85	0.72	0.83	0.73
ResearchAssistant	0.96	0.94	0.94	0.94	0.94	0.98	1.00	0.98	0.98	0.96	0.96
TeachingAssistant	0.96	0.96	0.94	0.94	0.94	0.98	0.98	0.98	0.98	0.94	0.96
Professor	0.93	0.68	0.76	0.83	0.87	0.93	0.96	0.76	0.68	0.62	0.68
Room	0.94	1.00	1.00	1.00	1.00	1.00	0.92	0.94	0.90	0.93	0.98
PhDStudent	0.97	0.78	0.88	0.32	0.79	0.97	1.00	0.64	0.95	0.82	0.78
Lecture	0.86	0.95	0.95	0.95	0.95	0.95	1.00	0.90	1.00	0.97	0.93
FullProfessor	0.95	0.81	0.92	0.92	0.90	0.95	0.97	0.84	0.84	0.89	0.82
AssociateProfessor	0.94	0.57	0.90	0.79	0.89	0.96	0.96	0.79	0.68	0.90	0.80
Staff	1.00	0.82	0.94	0.59	0.52	1.00	1.00	0.78	0.95	0.88	0.87

In summary, it seems that the two measures either complement each other or agree. The two former aggregations heavily depend on the higher or the lower aggregated value, while the two latter aggregations are rather an average between these values. In spite of the different techniques used, they provide similar results.

Solution D.8 (Thresholds) The correspondences resulting from the application of a .6 threshold to the similarity of Problem C.8 are:

1. for a hard threshold of .6, these are the following (11):

$$\begin{array}{ll}
 \text{PhDStudent} = .92 \text{ Student} & \text{Staff} = .65 \text{ Student} \\
 \text{PhDStudent} = .65 \text{ Philosophy} & \text{Staff} = .64 \text{ Science} \\
 \text{PhDStudent} = .64 \text{ People} & \text{Staff} = .63 \text{ Staff} \\
 \text{Professor} = .62 \text{ People} & \text{Assistant} = .62 \text{ Science} \\
 \text{Room} = .62 \text{ Office} & \text{Professor} = .60 \text{ Faculty} \\
 \text{PhDStudent} = .60 \text{ Science}
 \end{array}$$

3. for a proportional threshold of .6, they are those over the best value (.92) multiplied by .6, i.e., those over .552, that are the previous ones plus the following 2 correspondences (+2 = 13):

$$\text{Assistant} = .58 \text{ Faculty} \quad \text{Staff} = .56 \text{ People}$$

2. for a delta threshold of .6, they are those over the best value (.92) minus .6, i.e., those over .32, that is the previous ones plus 17 correspondences ($+17 = 30$):

Professor = .55 Staff	PhDStudent = .55 Staff
PhDStudent = .52 Boxology	Assistant = .46 Philosophy
PhDStudent = .45 Faculty	Professor = .44 Philosophy
Assistant = .44 Student	Assistant = .43 Staff
Professor = .40 Science	Assistant = .40 People
Professor = .36 Student	Professor = .36 Office
Lecture = .34 Philosophy	PhDStudent = .34 Office
Assistant = .33 Boxology	Staff = .33 Faculty
PhDStudent = .33 Course	

4. for a percentage threshold of .6, they are the 60 % best correspondences, i.e., the 37 best correspondences, that is the previous ones plus 7 correspondences ($+7 = 37$ because there are three correspondence at .23):

Professor = .32 Boxology	Assistant = .32 Course
Reference = .28 Boxology	Reference = .26 Philosophy
Lecture = .26 Faculty	Reference = .25 Science
Reference = .23 People	

... or any of the other at .23

Solution D.9 (Alignment extraction) Consider the two ontologies o and o' of Problem C.1 as illustrated in Fig. C.1; assume that the similarity between their entities is expressed by the similarity table of Problem C.8.

1. The alignment extracted with the greedy algorithm, is as follows and generate a stable marriage:

PhDStudent = .92 Student	Staff = .64 Science
Professor = .62 People	Room = .62 Office
Assistant = .58 Faculty	Lecture = .34 Philosophy
Reference = .28 Boxology	

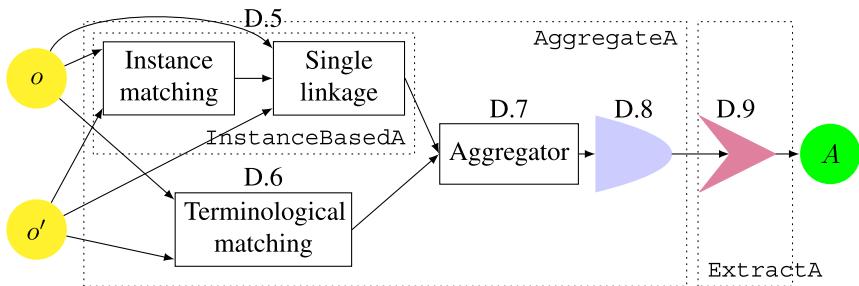
2. The alignment provided by the greedy algorithm is also a pairwise maximal matching;

3. The weights of the alignment extracted by the greedy algorithms sum to 4.0. The maximum weight alignment, given below, has weights summing to 4.03:

$\text{PhDStudent} = .92 \text{ Student}$	$\text{Staff} = .63 \text{ Staff}$
$\text{Professor} = .62 \text{ People}$	$\text{Room} = .62 \text{ Office}$
$\text{Assistant} = .62 \text{ Science}$	$\text{Lecture} = .34 \text{ Philosophy}$
$\text{Reference} = .28 \text{ Boxology}$	

Solution D.10 (Composing matchers) Consider the application described in Solution D.1.

1. The matching architecture developed so far would be:



The advantage of this architecture is that it allows for testing many approaches for terminological matching (D.6), instance-based matching (D.5), aggregation (D.7), applying threshold (D.8) or extracting (D.9) the alignment.

2. The figure above provides the names of the classes, available on the book web site, that implement the surrounding dotted composition. The implementation in a new Alignment class corresponds to composing the AggregateAlignment and ExtractAlignment processes. As implemented in the ComposedAlignment class, this can be achieved through:

```

public void align( Alignment alignment, Properties params )
    throws AlignmentException {
    // Call AggregateAlignment
    matcher.align( alignment, params );

    // Call ExtractAlignment... that only do extraction from matcher
    extractor.align( matcher, params );

    // Extract alignment
    for ( Cell c : extractor ) { addCell( c ); }
}
  
```

D.6 Evaluation of Matching Systems

Solution D.11 (Precision and recall computation) The precision, recall, F-measure, overall and Hamming distance alignments A_1 , A_2 , A_3 and R are given in the following table:

Alignment	precision	recall	F-measure	overall	Hamming distance
R	1.	1.	1.	1.	1.
A_1	.33	.33	.33	-.33	.2
A_2	.66	.66	.66	.33	.5
A_3	.33	.33	.33	-.33	.2

Hence A_2 is the best alignment. The results for A_3 are different from those given by the Alignment API because we have considered here that the correspondence Student = PhDStudent is not correct because of the relation, which should have been \geq , though the Alignment API only considers the entities and thus considers the correspondence correct (all values are .5).

Solution D.12 (Application specific evaluation) We have three systems and only one application. This application, given our previous answers speed (run time = No) and automation (automatic = NO) are not important, precision is very important (correct = YES), recall moderately important (complete = Yes). Hence, we would assign weights 1 to speed and automation, 5 to precision, 4 to recall, which, once normalised, yield .1, .5 and .4. Since we have no information about speed, it is not taken into account and all systems are equally automated.

			A_1	A_2	A_3
	Automatic	Precision			
	Automatic	Precision	1.	1.	1.
		Recall	.33	.66	.33
Our application	.1	.5	.4	.27	.63

The results are very similar to those provided by F-measure.

D.7 Representing Alignments

Solution D.13 (Representation generation) Consider R as described in Fig. C.2.

1. R in OWL as given by the Alignment API (see Solution D.15, Question 1, for the actual API output):

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY admin "http://book.ontologymatching.org/exercise/admin.owl#" >
  <!ENTITY lab   "http://book.ontologymatching.org/exercise/lab.owl#" >
>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<owl:Ontology rdf:about="">
  <rdfs:comment>OWL Alignment rendering</rdfs:comment>
  <rdfs:comment>(altered from OWLAxiomsRendererVisitor
                output)</rdfs:comment>
</owl:Ontology>

<owl:Class rdf:about="#admin;Office">
  <owl:equivalentClass rdf:resource="#lab;Room"/>
</owl:Class>
<owl:Class rdf:about="#lab;PhDStudent">
  <rdfs:subClassOf rdf:resource="#admin;Student"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#admin;room">
  <owl:equivalentProperty rdf:resource="#lab;office"/>
</owl:ObjectProperty>
<owl:Class rdf:about="#admin;Faculty">
  <owl:equivalentClass rdf:resource="#lab;FullProfessor"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#admin;teaches">
  <owl:equivalentProperty rdf:resource="#lab;teaches"/>
</owl:ObjectProperty>
<owl:Class rdf:about="#admin;Course">
  <owl:equivalentClass rdf:resource="#lab;Lecture"/>
</owl:Class>
</rdf:RDF>

```

2. *R* in C-OWL as given by the Alignment API:

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY admin "http://book.ontologymatching.org/exercise/admin.owl#" >
  <!ENTITY lab   "http://book.ontologymatching.org/exercise/lab.owl#" >
>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:cowl="http://www.itc.it/cowl#"
  xmlns:base="http://www.itc.it/cowl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<cowl:Mapping rdf:ID="">
  <cowl:sourceOntology>
    <owl:Ontology
      rdf:about="http://book.ontologymatching.org/exercise/admin.owl"/>
  </cowl:sourceOntology>
  <cowl:targetOntology>
    <owl:Ontology
      rdf:about="http://book.ontologymatching.org/exercise/lab.owl"/>
  </cowl:targetOntology>
  <cowl:bridgeRule>
    <cowl:Equivalent>
      <cowl:source>
        <owl:Class rdf:about="#admin;Office"/>
      </cowl:source>
      <cowl:target>

```

```

        <owl:Class rdf:about="#lab;Room" />
      </cowl:target>
    </cowl:Equivalent>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Onto>
      <cowl:source>
        <owl:Class rdf:about="#admin;Student" />
      </cowl:source>
      <cowl:target>
        <owl:Class rdf:about="#lab;PhDStudent" />
      </cowl:target>
    </cowl:Onto>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Equivalent>
      <cowl:source>
        <owl:ObjectProperty rdf:about="#admin;room" />
      </cowl:source>
      <cowl:target>
        <owl:ObjectProperty rdf:about="#lab;office" />
      </cowl:target>
    </cowl:Equivalent>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Equivalent>
      <cowl:source>
        <owl:Class rdf:about="#admin;Faculty" />
      </cowl:source>
      <cowl:target>
        <owl:Class rdf:about="#lab;FullProfessor" />
      </cowl:target>
    </cowl:Equivalent>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Equivalent>
      <cowl:source>
        <owl:ObjectProperty rdf:about="#admin;teaches" />
      </cowl:source>
      <cowl:target>
        <owl:ObjectProperty rdf:about="#lab;teaches" />
      </cowl:target>
    </cowl:Equivalent>
  </cowl:bridgeRule>
  <cowl:bridgeRule>
    <cowl:Equivalent>
      <cowl:source>
        <owl:Class rdf:about="#admin;Course" />
      </cowl:source>
      <cowl:target>
        <owl:Class rdf:about="#lab;Lecture" />
      </cowl:target>
    </cowl:Equivalent>
  </cowl:bridgeRule>
</cowl:Mapping>
</rdf:RDF>

```

3. R in SWRL as given by the Alignment API (this generates 11 rules):

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE swrlx:Ontology [
  <!ENTITY admin "http://book.ontologymatching.org/exercise/admin.owl#" >
  <!ENTITY lab   "http://book.ontologymatching.org/exercise/lab.owl#" >
>

<swrlx:Ontology swrlx:name="generatedA1"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

```

```

xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
xmlns:owlx="http://www.w3.org/2003/05/owl-xml"
xmlns:ruleml="http://www.w3.org/2003/11/ruleml#">

<!-- Generated by fr.inrialpes.exmo.impl.renderer.SWRLRendererVisitor -->
<owlx:Annotation>
  <owlx:Documentation>method: Manually generated
    (Jérôme Euzenat, 2008/04/17)</owlx:Documentation>
</owlx:Annotation>

<owlx:Imports
  rdf:resource="http://book.ontologymatching.org/exercise/admin.owl"/>

<ruleml:imp>
  <ruleml:_body>
    <swrl:classAtom>
      <owlx:Class owlx:name="&admin;Office" />
      <ruleml:var>x</ruleml:var>
    </swrl:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&lab;Room" />
      <ruleml:var>x</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>

<ruleml:imp>
  <ruleml:_body>
    <swrl:classAtom>
      <owlx:Class owlx:name="&lab;Room" />
      <ruleml:var>x</ruleml:var>
    </swrl:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&admin;Office" />
      <ruleml:var>x</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>

<ruleml:imp>
  <ruleml:_body>
    <swrl:classAtom>
      <owlx:Class owlx:name="&admin;Course" />
      <ruleml:var>x</ruleml:var>
    </swrl:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&lab;Lecture" />
      <ruleml:var>x</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>

<ruleml:imp>
  <ruleml:_body>
    <swrl:classAtom>
      <owlx:Class owlx:name="&lab;Lecture" />
      <ruleml:var>x</ruleml:var>
    </swrl:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>

```

```

<owl:Class owl:label="admin;Course"/>
<ruleml:var>x</ruleml:var>
</swrl:classAtom>
</ruleml:_head>
</ruleml:imp>

<ruleml:imp>
<ruleml:_body>
<swrl:classAtom>
<owl:Class owl:label="admin;Faculty"/>
<ruleml:var>x</ruleml:var>
</swrl:classAtom>
</ruleml:_body>
<ruleml:_head>
<swrlx:classAtom>
<owl:Class owl:label="lab;FullProfessor"/>
<ruleml:var>x</ruleml:var>
</swrlx:classAtom>
</ruleml:_head>
</ruleml:imp>

<ruleml:imp>
<ruleml:_body>
<swrl:classAtom>
<owl:Class owl:label="lab;FullProfessor"/>
<ruleml:var>x</ruleml:var>
</swrl:classAtom>
</ruleml:_body>
<ruleml:_head>
<swrlx:classAtom>
<owl:Class owl:label="admin;Faculty"/>
<ruleml:var>x</ruleml:var>
</swrlx:classAtom>
</ruleml:_head>
</ruleml:imp>

<ruleml:imp>
<ruleml:_body>
<swrl:individualPropertyAtom swrlx:property="admin;teaches"/>
<ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
</swrl:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head>
<swrl:individualPropertyAtom swrlx:property="lab;teaches"/>
<ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
</swrl:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>

<ruleml:imp>
<ruleml:_body>
<swrl:individualPropertyAtom swrlx:property="lab;teaches"/>
<ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
</swrl:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head>
<swrl:individualPropertyAtom swrlx:property="admin;teaches"/>
<ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
</swrl:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>

<ruleml:imp>
<ruleml:_body>
<swrl:classAtom>
<owl:Class owl:label="lab;PhDStudent"/>
<ruleml:var>x</ruleml:var>

```

```

        </swrl:classAtom>
    </ruleml:_body>
    <ruleml:_head>
        <swrlx:Class owl:Class name="&admin;Student"/>
        <ruleml:var>x</ruleml:var>
    </swrl:classAtom>
    </ruleml:_head>
</ruleml:imp>

<ruleml:imp>
    <ruleml:_body>
        <swrl:individualPropertyAtom swrlx:property="&admin;room"/>
        <ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
    </swrl:individualPropertyAtom>
    </ruleml:_body>
    <ruleml:_head>
        <swrl:individualPropertyAtom swrlx:property="&lab;office"/>
        <ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
    </swrl:individualPropertyAtom>
    </ruleml:_head>
</ruleml:imp>

<ruleml:imp>
    <ruleml:_body>
        <swrl:individualPropertyAtom swrlx:property="&lab;office"/>
        <ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
    </swrl:individualPropertyAtom>
    </ruleml:_body>
    <ruleml:_head>
        <swrl:individualPropertyAtom swrlx:property="&admin;room"/>
        <ruleml:var>x</ruleml:var><ruleml:var>y</ruleml:var>
    </swrl:individualPropertyAtom>
    </ruleml:_head>
</ruleml:imp>

</swrlx:Ontology>

```

4. R in the EDOAL mapping language as given by the Alignment API:

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE swrlx:Ontology [
    <!ENTITY admin "http://book.ontologymatching.org/exercise/admin.owl#" >
    <!ENTITY lab   "http://book.ontologymatching.org/exercise/lab.owl#" >
    <!ENTITY xsd   "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF
    xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema#'
    xmlns:align='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
    xmlns:edoal='http://ns.inria.org/edoal/1.0/#'>

<Alignment>
    <xml>yes</xml>
    <level>2EDOAL</level>
    <type>**</type>
    <method>fr.inrialpes.exmo.align.edoal.EDOALAlignment#toEDOAL</method>
    <onto1>
        <Ontology
            rdf:about="http://book.ontologymatching.org/exercise/admin.owl">
            <location>http://book.ontologymatching.org/exercise/admin.owl</location>
            <formalism>
                <Formalism align:name="OWL2.0"
                    align:uri="http://www.w3.org/2002/07/owl#" />
            </formalism>
        </Ontology>
    </onto1>

```

```

<onto2>
  <Ontology rdf:about="http://book.ontologymatching.org/exercise/lab.owl">
    <location>http://book.ontologymatching.org/exercise/lab.owl</location>
    <formalism>
      <Formalism align:name="OWL2.0"
                  align:uri="http://www.w3.org/2002/07/owl#" />
    </formalism>
  </Ontology>
</onto2>
<map>
  <Cell>
    <entity1><edoal:Relation rdf:about="#admin;room"/></entity1>
    <entity2><edoal:Relation rdf:about="#lab;office"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1><edoal:Class rdf:about="#admin;Office"/></entity1>
    <entity2><edoal:Class rdf:about="#lab;Room"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1><edoal:Class rdf:about="#admin;Faculty"/></entity1>
    <entity2><edoal:Class rdf:about="#lab;FullProfessor"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1><edoal:Class rdf:about="#admin;Student"/></entity1>
    <entity2><edoal:Class rdf:about="#lab;PhDStudent"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1><edoal:Relation rdf:about="#admin;teaches"/></entity1>
    <entity2><edoal:Relation rdf:about="#lab;teaches"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1><edoal:Class rdf:about="#admin;Course"/></entity1>
    <entity2><edoal:Class rdf:about="#lab;Lecture"/></entity2>
    <relation>=</relation>
    <measure rdf:datatype='xsd:float'>1.0</measure>
  </Cell>
</map>
</Alignment>
</rdf:RDF>

```

5. R in SKOS as given by the Alignment API:

```

<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY admin "http://book.ontologymatching.org/exercise/admin.owl#" >
  <!ENTITY lab   "http://book.ontologymatching.org/exercise/lab.owl#" >
>
<rdf:RDF>

```

```


xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:skos="http://www.w3.org/2008/05/skos#"

<skos:Concept rdf:about="#admin;Office">
  <skos:exactMatch rdf:resource="#lab;Room"/>
</skos:Concept>

<skos:Concept rdf:about="#admin;room">
  <skos:exactMatch rdf:resource="#lab;office"/>
</skos:Concept>

<skos:Concept rdf:about="#admin;Student">
  <skos:broadMatch rdf:resource="#lab;PhDStudent"/>
</skos:Concept>

<skos:Concept rdf:about="#admin;Course">
  <skos:exactMatch rdf:resource="#lab;Lecture"/>
</skos:Concept>

<skos:Concept rdf:about="#admin;teaches">
  <skos:exactMatch rdf:resource="#lab;teaches"/>
</skos:Concept>

<skos:Concept rdf:about="#admin;Faculty">
  <skos:exactMatch rdf:resource="#lab;FullProfessor"/>
</skos:Concept>

</rdf:RDF>

```

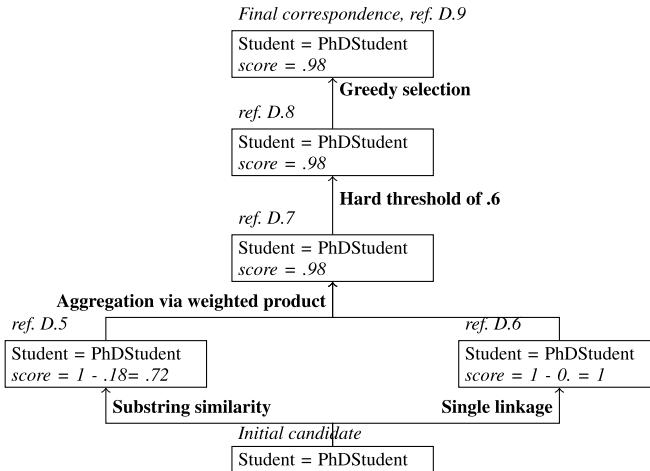
6. The discussion on the advantages of alignment formats is rather generic. The OWL representation provides a merge between the two ontologies. Once loaded, this OWL does not distinguish an alignment and two ontologies, but only one ontology. The other representations maintain the two ontologies and represent the alignments as mappings (C-OWL, EDOAL or SKOS) or rules (SWRL). SWRL and C-OWL use oriented mappings (from source to target) while the others are not oriented. OWL, C-OWL and SWRL are immediately operational for reasoning and expressed in a formally defined language, while EDOAL and SKOS require a processor. With respect to the RDF format, all these representations, but EDOAL, lose information, such as the metadata on the alignments or the confidence measures.

D.8 Explaining Alignments

Solution D.14 (Alignment explanation) Consider the ontologies of Fig. C.2 and the architecture provided in Solution D.10.

1. A decision flow for the correspondence `Student = PhDStudent` is provided below. Following the discussion of Solution D.11, the relation of this correspondence is not specific enough, and hence, it is an incorrect one.
The above presented decision flow would be that of a matching system able to compute only equivalence correspondences. However, a system able to compute subsumption relations, e.g., through a common suffix rule (Sect. 5.2.1), hence being able to generate the `Student ≥ PhDStudent` as an initial candidate, would

manage to deliver it as the final and correct correspondence with the same explanation flow.



- Both the terminological substring matcher and the instance-based single linkage matcher agree that there is a relation between entities Student and PhDStudent by assigning them a high similarity, which was further reinforced through an aggregation via weighted product. This correspondence, with the similarity of .98, passed the hard threshold of .6 and was also retained by the greedy alignment extraction algorithm (and is even part of the maximal weight alignment). Hence, it was delivered as the final one. The (in)correctness depends of the ability of a matching system to compute a specialisation relation.

D.9 Processing Alignments

Solution D.15 (Merging ontologies)

- The merge between o and o' according to R can be expressed with OWL import by:

```

<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" 
           rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
           xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<owl:Ontology rdf:about="">
  <rdfs:comment>Matched ontologies</rdfs:comment>
  <owl:imports
    rdf:resource="http://book.ontologymatching.org/exercise/admin.owl"/>
  <owl:imports
    rdf:resource="http://book.ontologymatching.org/exercise/lab.owl"/>
</owl:Ontology>

<owl:Class
  rdf:about="http://book.ontologymatching.org/exercise/admin.owl#Office">

```

```

<owl:equivalentClass
  rdf:resource="http://book.ontologymatching.org/exercise/lab.owl#Room"/>
</owl:Class>
<owl:Class
  rdf:about="http://book.ontologymatching.org/exercise/lab.owl#PhDStudent">
  <rdfs:subClassOf
    rdf:resource="http://book.ontologymatching.org/exercise/admin.owl#Student"/>
</owl:Class>
<owl:ObjectProperty
  rdf:about="http://book.ontologymatching.org/exercise/admin.owl#room">
  <owl:equivalentProperty
    rdf:resource="http://book.ontologymatching.org/exercise/lab.owl#office"/>
</owl:ObjectProperty>
<owl:Class
  rdf:about="http://book.ontologymatching.org/exercise/admin.owl#Faculty">
  <owl:equivalentClass rdf:resource=
    "http://book.ontologymatching.org/exercise/lab.owl#FullProfessor"/>
</owl:Class>
<owl:ObjectProperty
  rdf:about="http://book.ontologymatching.org/exercise/admin.owl#teaches">
  <owl:equivalentProperty
    rdf:resource="http://book.ontologymatching.org/exercise/lab.owl#teaches"/>
</owl:ObjectProperty>
<owl:Class
  rdf:about="http://book.ontologymatching.org/exercise/admin.owl#Course">
  <owl:equivalentClass
    rdf:resource="http://book.ontologymatching.org/exercise/lab.owl#Lecture"/>
</owl:Class>
</rdf:RDF>
```

This is what is indeed rendered by the `OWLAxiomsRendererVisitor` of the Alignment API.

2. It can also be expressed as an integrated OWL ontology like the one of Fig. D.3, in which Faculty replaces FullProfessor, Staff in o has been renamed Staff#2, Room replaces Office, Lecture replaces Course, office replaces room and both teaches have been unified.

Solution D.16 (Data translation) Assume that one wants to transform the data instances from ontology o into instances of o' according to the reference alignment R as described in Fig. C.2. Consider the data instances in the first table of Problem C.6.

1. The transformation of data instances in the first table of Problem C.6 with regard to the reference alignment R would lead to the creation of six Full professors with no name, no hiring date and no office. Indeed, two Students had not sufficient information to convert them into PhDStudents and one Staff instance has no corresponding class in the alignment. This means that the alignment does not contain sufficient information to convert all data. However, it is already useful by identifying six FullProfessor.
2. There may be several ways to perform the translation using the functions of the Alignment API. If the data is or can be expressed in XML, then the Alignment API is able to generate an XSLT transformation thanks to its `XSLTRendererVisitor` serialiser. This transformation may then be applied to the XML data. If the data is available behind a SPARQL endpoint, it is possible to use the `SPARQLConstructRendererVisitor` serialiser which extracts triples and

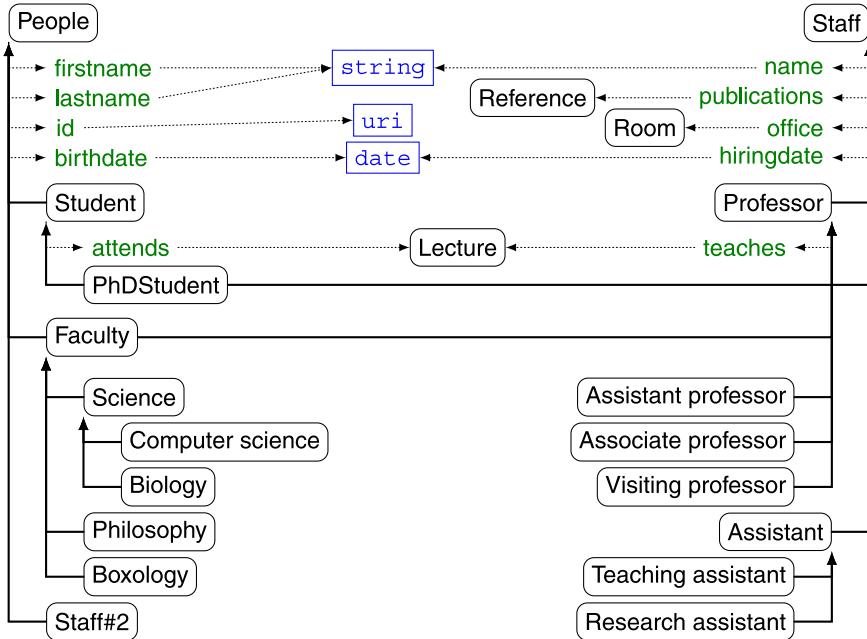


Fig. D.3 Merge of the ontology o and o' with regard to the reference alignment R .

generates new triples. Otherwise, it is necessary to use the alignment from Java for performing the adequate actions. Here is the core of a program that does this with the Alignment API and Jena:

```

public void transform( OntModel onto1, OntModel onto2, Alignment al ) {
    Hashtable<String,Individual> ht = new Hashtable<String,Individual>();
    URI base = al.getOntology2URI();
    // For each correspondence involving a class, transform individuals
    for ( Cell c : al ) {
        OntClass cl = onto1.getOntClass( c.getObject1AsURI( al ).toString() );
        if ( cl != null && ( c.getRelation().getRelation().equals(">")
                || c.getRelation().getRelation().equals("<") ) ) {
            OntClass c12 = onto2.getOntClass(
                c.getObject2AsURI( al ).toString() );
            for ( Individual i : listInstances( cl ) ) {
                String id = base.getScheme()+base.getSchemeSpecificPart()
                    +URLEncoder.encode( i.getLabel(""), "UTF-8" );
                Individual o = onto2.createIndividual( id, c12 );
                ht.put( i.getURI(), o );
                System.out.println(o.getURI()+" rdf:type "+o.getOntClass()+" .");
            }
        }
    }
    // For each correspondence involving a property, transform statements
    for ( Cell c : al ) {
        OntProperty pr =
            onto1.getOntProperty( c.getObject1AsURI( al ).toString() );
        if ( pr != null ) {
            OntProperty pr2 =
                onto2.getOntProperty( c.getObject2AsURI( al ).toString() );
            StmtIterator stmtit =

```

```

onto1.listStatements( (Resource)null, pr, (RDFNode)null );
for ( ; stmtit.hasNext() ; ) {
    Statement st = stmtit.nextStatement();
    Individual subj = (Individual)ht.get( st.getSubject().getURI() );
    if ( subj != null ) {
        RDFNode obj = st.getObject();
        if ( obj.isResource() ) {
            obj = (Individual)ht.get( obj.asResource().getURI() );
        }
        if ( obj != null && !obj.isAnon() ) { // do not translate
            onto2.createStatement( subj, pr2, obj );
            System.out.println( subj+" "+pr2+" "+obj+" ." );
        }
    }
}
}

```

3. Data interlinking may be used for first identifying resources expressed in both ontologies before applying the transformation. This would complete the existing identified resources instead of generating brand new disconnected and incomplete ones. In this case, a simple measure would be to consider that, if the name of a Staff member of o' starts with the firstname and ends with the lastname of some People of o , then they can be presumed to be the same person. This raises the problem that Paola and Pierluigi Pomodoro in o' may be matched to the same P. Pomodoro of o , so it is necessary to be cautious. But it also helps in identifying that four out of the six initial FullProfessors were already existing in o' . In addition, Stefano Zucchini can now be identified as a PhDStudent which is already existing on o' so his information is not duplicated.

Solution D.17 (Mediation)

- ### 1. The query:

```
SELECT ?room  
WHERE {  
    ?x rdf:type o:Faculty .  
    ?x o:room ?room .  
}
```

expressed with respect to ontology \mathcal{O} involves finding the correspondences for attribute room and class Faculty. These are respectively office and FullProfessor. Hence the translated query by the reference alignment R as described in Fig. C.2 is:

```
SELECT ?room
WHERE {
    ?x rdf:type o':FullProfessor .
    ?x o':office ?room .
}
}
```

- When evaluated against the second table, the answer will be $\{\langle B12 \rangle\}$ because the only instance of the class FullProfessor in the second ontology is Federico di Guava whose office is B12 (or rather the room identified by B12).
 - A mediator able to perform the transformations of any query q based on the reference alignment R will have to transform the query with regard to R ($q' =$

$\text{TransformQuery}(q, R)$), it will then have to evaluate the result ($r' = \text{Eval}(q')$) and to return the results by translating them with regard to the converse of R ($r = \text{Translate}(r', \text{Invert}(R))$). Hence, the full mediator would be implemented by:

$$\text{Translate}(\text{Eval}(\text{TransformQuery}(q, R)), \text{Invert}(R))$$

References

- Aberer, K., Catarci, T., Cudré-Mauroux, P., Dillon, T., Grimm, S., Hacid, M.-S., Illarramendi, A., Jarrar, M., Kashyap, V., Mecella, M., Mena, E., Neuhold, E., Ouksel, A., Risse, T., Scannapieco, M., Saltor, F., De Santis, L., Spaccapietra, S., Staab, S., Studer, R., De Troyer, O.: Emergent semantics systems. In: Proc. 1st International Conference on Semantics of a Networked World (ICSNW), Paris, France. Lecture Notes in Computer Science, vol. 3226, pp. 14–43 (2004a) (p. 15)
- Aberer, K., Cudré-Mauroux, P., Ouksel, A., Catarci, T., Hacid, M.-S., Illarramendi, A., Kashyap, V., Mecella, M., Mena, E., Neuhold, E., De Troyer, O., Risse, T., Scannapieco, M., Saltor, F., de Santis, L., Spaccapietra, S., Staab, S., Studer, R.: Emergent semantics principles and issues. In: Proc. 9th International Conference on Database Systems for Advanced Applications (DAS-FAA), Jeju Island, Korea. Lecture Notes in Computer Science, vol. 2973, pp. 25–38 (2004b) (p. 15)
- Agrawal, R., Srikant, R.: On integrating catalogs. In: Proc. 10th International World Wide Web Conference (WWW), Hong Kong, China, pp. 603–612 (2001) (p. 6)
- Aguirre, J.L., Eckert, K., Euzenat, J., Ferrara, A., van Hage, W.R., Hollink, L., Meilicke, C., Nikolov, A., Ritze, D., Scharffé, F., Shvaiko, P., Šváb-Zamazal, O., Trojahn, C., Jiménez-Ruiz, E., Cuena Grau, B., Zapilko, B.: Results of the ontology alignment evaluation initiative 2012. In: Proc. 7th International Workshop on Ontology Matching (OM) at the 11th International Semantic Web Conference (ISWC), Boston, MA, USA, pp. 73–115 (2012) (pp. 62, 289)
- Alagic, S., Bernstein, P.: A model theory for generic schema management. In: Proc. 8th International Workshop on Database Programming Languages (DBPL), Frascati, Italy. Lecture Notes in Computer Science, vol. 2397, pp. 228–246 (2001) (p. 49)
- Albagli, S., Ben-Eliyahu-Zohary, R., Shimony, S.E.: Markov network based ontology matching. *J. Comput. Syst. Sci.* **78**(1), 105–118 (2012) (pp. 143, 233)
- Alexe, B., Chiticariu, L., Miller, R., Tan, W.C.: Muse: Mapping Understanding and deSign by Example. In: Proc. 24th International Conference on Data Engineering (ICDE), Cancún, Mexico, pp. 10–19 (2008a) (p. 371)
- Alexe, B., Tan, W.C., Velegrakis, Y.: Comparing and evaluating mapping systems with STBenchmark. *Proc. VLDB Endow.* **1**(2), 1468–1471 (2008b) (p. 299)
- Alexe, B., Tan, W.C., Velegrakis, Y.: STBenchmark: towards a benchmark for mapping systems. *Proc. VLDB Endow.* **1**(1), 230–244 (2008c) (p. 299)
- Algergawy, A., Nayak, R., Siegmund, N., Köppen, V., Saake, G.: Combining schema and level-based matching for web service discovery. In: Proc. 10th International Conference on Web Engineering (ICWE), Vienna, Austria, pp. 114–128 (2010) (pp. 16, 219)
- Amgoud, L., Parsons, S., Maudet, N.: Arguments, dialogue and negotiation. In: Proc. 14th European Conference on Artificial Intelligence (ECAI), Berlin, Germany, pp. 338–342 (2000) (p. 171)

- An, Y., Borgida, A., Mylopoulos, J.: Constructing complex semantic mappings between XML data and ontologies. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 6–20 (2005) (p. 213)
- An, Y., Borgida, A., Mylopoulos, J.: Discovering the semantics of relational tables through mappings. *J. Data Semant.* **VII**, 1–32 (2006) (p. 213)
- Araújo, S., Tran, D., DeVries, A., Hidders, J., Schwabe, D.: SERIMI: class-based disambiguation for effective instance matching over heterogeneous web data. In: Proc. 15th International Workshop on the Web and Databases (WebDB) at the International Conference on Management of Data (SIGMOD), Scottsdale, AZ, USA, pp. 25–30 (2012) (p. 117)
- Arens, Y., Hsu, C.-N., Knoblock, C.: Query processing in the SIMS information mediator. In: Tate, A. (ed.) *Readings in Agents*, pp. 82–90. AAAI Press, Menlo Park (1996) (p. 201)
- Atencia, M., Schorlemmer, M.: An interaction-based approach to semantic alignment. *J. Web Semant.* **13**(1), 131–147 (2012) (pp. 15, 49)
- Atencia, M., Euzenat, J., Pirrò, G., Rousset, M.-C.: Alignment-based trust for resource finding in semantic P2P networks. In: Proc. 10th International Semantic Web Conference (ISWC), Bonn, Germany. Lecture Notes in Computer Science, vol. 7031, pp. 51–66 (2011) (p. 16)
- Atencia, M., Borgida, A., Euzenat, J., Ghidini, C., Serafini, L.: A formal semantics for weighted ontology mappings. In: Proc. 11th International Semantic Web Conference (ISWC), Boston, MA, USA. Lecture notes in computer science, vol. 7649, pp. 17–33 (2012a) (p. 44)
- Atencia, M., David, J., Scharffe, F.: Keys and pseudo-keys detection for web datasets cleansing and interlinking. In: Proc. 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Galway, Ireland. Lecture Notes in Computer Science, vol. 7603, pp. 144–153 (2012b) (p. 116)
- Atzeni, P., Cappellari, P., Bernstein, P.: ModelGen: model independent schema translation. In: Proc. 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, pp. 1111–1112 (2005) (p. 342)
- Atzeni, P., Cappellari, P., Bernstein, P.: Model-independent schema and data translation. In: Proc. 10th Conference on Extending Database Technology (EDBT), Munich, Germany. Lecture Notes in Computer Science, vol. 3896, pp. 368–385 (2006) (p. 342)
- Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: MISM: a platform for Model-Independent Solutions to Model management problems. *J. Data Semant.* **XIV**, 133–161 (2009) (p. 342)
- Aumüller, D., Do, H.-H., Maßmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proc. 24th International Conference on Management of Data (SIGMOD), Software Demonstration, Baltimore, MD, USA, pp. 906–908 (2005) (p. 355)
- Avesani, P., Giunchiglia, F., Yatskevich, M.: A large scale taxonomy mapping evaluation. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 67–81 (2005) (p. 297)
- Bach, T.-L., Dieng-Kuntz, R.: Measuring similarity of elements in OWL ontologies. In: Proc. 1st International Workshop on Contexts and Ontologies (C&O) at the 20th National Conference on Artificial Intelligence (AAAI), Pittsburgh, PA, USA, pp. 96–99 (2005) (p. 217)
- Bach, T.-L., Dieng-Kuntz, R., Gandon, F.: On ontology matching problems (for building a corporate semantic web in a multi-communities organization). In: Proc. 6th International Conference on Enterprise Information Systems (ICEIS), Porto, Portugal, pp. 236–243 (2004) (pp. 123, 217)
- Bailin, S., Truszkowski, W.: Ontology negotiation: how agents can really get to know each other. In: Proc. 1st International Workshop on Radical Agent Concepts (WRAC), McLean, VA, USA. Lecture Notes in Computer Science, vol. 2564, pp. 320–334 (2002) (p. 19)
- Bao, J., Voutsadakis, G., Slutzki, G., Honavar, V.: Package-based description logics. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, pp. 349–371. Springer, Berlin (2009) (p. 50)
- Barthélemy, J.-P., Guénoc'h, A.: *Trees and Proximity Representations*. Wiley, Chichester (1992) (p. 124)
- Barwise, J., Seligman, J.: *Information Flow: the Logic of Distributed Systems*. Cambridge Tracts in Theoretical Computer Science, vol. 44. Cambridge University Press, Cambridge (1997) (p. 243)

- Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.* **18**(4), 323–364 (1986) (pp. 5, 8, 37, 73, 228)
- Bechhofer, S., Volz, R., Lord, P.: Cooking the semantic web with the OWL API. In: Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, USA. Lecture Notes in Computer Science, vol. 2870, pp. 659–675 (2003) (p. 347)
- Bellahsene, Z., Bonifati, A., Rahm, E. (eds.): Schema Matching and Mapping. Data-Centric Systems and Applications. Springer, Berlin (2011) (pp. 39, 41, 73, 201)
- Bench-Capon, T.: Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.* **13**(3), 429–448 (2003) (p. 170)
- Bench-Capon, T., Malcolm, G.: Formalising ontologies and their relations. In: Proc. 16th International Conference on Database and Expert Systems Applications (DEXA), Copenhagen, Denmark. Lecture Notes in Computer Science, vol. 1677, pp. 250–259 (1999) (p. 49)
- Benerecetti, M., Bouquet, P., Ghidini, C.: Contextual reasoning distilled. *J. Exp. Theor. Artif. Intell.* **12**(3), 279–305 (2000) (p. 37)
- Benerecetti, M., Bouquet, P., Ghidini, C.: On the dimensions of context dependence: partiality, approximation, and perspective. In: Proc. 3rd International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT), Dundee, UK. Lecture Notes in Computer Science, vol. 2116, pp. 59–72 (2001) (p. 38)
- Beneventano, D., Bergamaschi, S., Lodi, S., Sartori, C.: Consistency checking in complex object database schemata with integrity constraints. *IEEE Trans. Knowl. Data Eng.* **10**(4), 576–598 (1998) (p. 206)
- Bergamaschi, S., Beneventano, D., Castano, S., Vincini, M.: MOMIS: an intelligent system for the integration of semistructured and structured data. Technical Report T3-R07, Università di Modena e Reggio Emilia, Modena, Italy (1998) (p. 206)
- Bergamaschi, S., Castano, S., Vincini, M.: Semantic integration of semistructured and structured data sources. *SIGMOD Rec.* **28**(1), 54–59 (1999) (p. 206)
- Berge, C.: Graphes et Hypergraphes. Dunod, Paris (1970) (p. 190)
- Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: Proc. 14th International Conference on Advanced Information Systems Engineering (CAiSE), Toronto, Canada. Lecture Notes in Computer Science, vol. 2348, pp. 452–466 (2002) (p. 237)
- Berners-Lee, T.: Linked-data design issues. W3C design issue document (2009). <http://www.w3.org/DesignIssues/LinkedData.html> (p. 11)
- Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. *Sci. Am.* **284**(5), 34–43 (2001) (p. 82)
- Bernstein, P., Rahm, E.: Data warehouse scenarios for model management. In: Proc. 19 International Conference on Conceptual Modeling (ER), Salt Lake City, UT, USA. Lecture Notes in Computer Science, vol. 1920, pp. 1–15 (2000) (pp. 5, 9)
- Bernstein, P., Halevy, A., Pottinger, R.: A vision of management of complex models. *SIGMOD Rec.* **29**(4), 55–63 (2000) (pp. 41, 341)
- Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: a vision. In: Proc. 5th International Workshop on the Web and Databases (WebDB), Madison, WI, USA (2002) (pp. 14, 15)
- Besana, P.: A framework for combining ontology and schema matchers with Dempster-Shafer. In: Proc. 1st International Workshop on Ontology Matching (OM) at the 5th International Semantic Web Conference (ISWC), Athens, GA, USA, pp. 196–200 (2006) (p. 169)
- Besana, P., Robertson, D.: How service choreography statistics reduce the ontology mapping problem. In: Proc. 6th International Semantic Web Conference (ISWC) and 2nd Asian Semantic Web Conference (ASWC), Busan, Korea. Lecture Notes in Computer Science, vol. 4825, pp. 44–57 (2007) (p. 264)
- Bilke, A., Naumann, F.: Schema matching using duplicates. In: Proc. 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, pp. 69–80 (2005) (pp. 174, 238)
- Birkens, D., Dodge, Y.: Alternative Methods of Regression. Wiley, New York (2001) (p. 183)
- Biron, P., Malhotra, A. (eds.): XML schema part 2: Datatypes. Recommendation, W3C (2004). <http://www.w3.org/TR/xpath> (p. 109)

- Bisson, G.: Learning in FOL with similarity measure. In: Proc. 10th National Conference on Artificial Intelligence (AAAI), San Jose, CA, USA, pp. 82–87 (1992) (p. 135)
- Bock, J., Hettenhausen, J.: Discrete particle swarm optimisation for ontology alignment. *Inf. Sci.* **192**, 152–173 (2012) (pp. 139, 231)
- Bock, J., Dänschel, C., Stumpp, M.: MapPSO and MapEVO results for OAEI 2011. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 179–183 (2011) (pp. 231, 232)
- Bodenreider, O.: The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Res.* **32**, 267–270 (2004) (p. 28)
- Boley, H., Kifer, M. (eds.): RIF basic logic dialect. Recommendation, W3C (2010). <http://www.w3.org/TR/rif-bl/> (p. 328)
- Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., Summa, G.: Schema mapping verification: the Spicy way. In: Proc. 11th International Conference on Extending Database Technology (EDBT), Nantes, France, pp. 85–96 (2008) (p. 384)
- Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading (1998) (p. 31)
- Borgida, A., Serafini, L.: Distributed description logics: assimilating information from peer sources. *J. Data Semant.* **I**, 153–184 (2003) (p. 50)
- Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: Proc. 5th Annual Conference on Computational Learning Theory (COLT), Pittsburgh, PA, USA, pp. 144–152 (1992) (p. 177)
- Bouquet, P., Magnini, B., Serafini, L., Zanobini, S.: A SAT-based algorithm for context matching. In: Proc. 4th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT), Stanford, CA, USA. Lecture Notes in Computer Science, vol. 2680, pp. 66–79 (2003a) (p. 213)
- Bouquet, P., Serafini, L., Zanobini, S.: Semantic coordination: a new approach and an application. In: Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, USA. Lecture Notes in Computer Science, vol. 2870, pp. 130–145 (2003b) (pp. 6, 8, 75, 213)
- Bouquet, P., Ehrig, M., Euzenat, J., Franconi, E., Hitzler, P., Krötzsch, M., Serafini, L., Stamou, G., Sure, Y., Tessaris, S.: Specification of a common framework for characterizing alignment. Deliverable D2.2.1, Knowledge web NoE (2004a). <http://knowledgeweb.semanticweb.org> (pp. 37, 38, 39, 41, 42)
- Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. *J. Web Semant.* **1**(1), 325–343 (2004b) (p. 326)
- Bouquet, P., Serafini, L., Zanobini, S., Sceffer, S.: Bootstrapping semantics on the web: meaning elicitation from schemas. In: Proc. 15th International World Wide Web Conference (WWW), Edinburgh, UK, pp. 505–512 (2006) (pp. 147, 213)
- Bourgault, D., Jacquemin, C.: Term extraction + term clustering: an integrated platform for computer-aided terminology. In: Proc. European Chapter of the Association for Computational Linguistics (EACL), Bergen, Norway, pp. 15–22 (1999) (p. 98)
- Boyd, M., Kittivoravitkul, S., Lazaridis, C., McBrien, P., Rizopoulos, N.: AutoMed: a BAV data integration system for heterogeneous data sources. In: Proc. 16th International Conference on Advanced Information Systems Engineering (CAiSE), Riga, Latvia. Lecture Notes in Computer Science, vol. 3084, pp. 82–97 (2004) (p. 201)
- Brasethvik, T., Gulla, J.A.: Natural language analysis for semantic document modeling. *Data Knowl. Eng.* **38**(1), 45–62 (2001) (p. 251)
- Breiman, L.: Stacked regressions. *Mach. Learn.* **24**(1), 49–64 (1996) (p. 183)
- Breitbart, Y.: Multidatabase interoperability. *SIGMOD Rec.* **19**(3), 53–60 (1990) (p. 37)
- Brill, E.: A simple rule-based part of speech tagger. In: Proc. 3rd Conference on Applied Natural Language Processing (ANLP), Trento, Italy, pp. 152–155 (1992) (p. 97)
- Brodie, M., Mylopoulos, J., Schmidt, J.: On Conceptual Modeling. Springer, New York (1984) (pp. vii, 31)
- Budanitsky, A., Hirst, G.: Evaluating WordNet-based measures of lexical semantic relatedness. *Comput. Linguist.* **32**(1), 13–47 (2006) (pp. 103, 106)

- Bussler, C., Fensel, D., Mädche, A.: A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.* **31**(4), 24–29 (2002) (p. 17)
- Cafarella, M., Halevy, A., Madhavan, J.: Structured data on the web. *Commun. ACM* **54**(2), 72–79 (2011) (p. 23)
- Calvanese, D., De Giacomo, G., Lenzerini, M.: Description logics for information integration. In: Kakas, A., Sadri, F. (eds.) *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*. Lecture Notes in Computer Science, vol. 2408, pp. 41–60. Springer, Heidelberg (2002a) (p. 201)
- Calvanese, D., Giacomo, G.D., Lenzerini, M.: A framework for ontology integration. In: Cruz, I., Decker, S., Euzenat, J., McGuinness, D. (eds.) *The Emerging Semantic Web*, pp. 201–214. IOS Press, Amsterdam (2002b) (p. 50)
- Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: Proc. 23rd Symposium on Principles of Database Systems (PODS), Paris, France, pp. 241–251 (2004) (p. 50)
- Caracciolo, C., Euzenat, J., Hollink, L., Ichise, R., Isaac, A., Malaisé, V., Meilicke, C., Pane, J., Shvaiko, P., Stuckenschmidt, H., Šváb-Zamazal, O., Svátek, V.: Results of the ontology alignment evaluation initiative 2008. In: Proc. 3rd International Workshop on Ontology Matching (OM) at the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany, pp. 73–119 (2008) (p. 357)
- Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proc. 13th International World Wide Web (WWW) Conference, New York, NY, USA, pp. 74–83 (2004) (p. 225)
- Castano, S., De Antonellis, V., De Capitani di Vimercati, S.: Global viewing of heterogeneous data sources. *IEEE Trans. Knowl. Data Eng.* **13**(2), 277–297 (2000) (p. 206)
- Castano, S., Ferrara, A., Montanelli, S.: Dynamic knowledge discovery in open, distributed and multi-ontology systems: techniques and applications. In: Taniar, D., Rahayu, J. (eds.) *Web Semantics and Ontology*, pp. 226–258. Idea Group, Hershey (2005). Chap. 8 (p. 207)
- Castano, S., Ferrara, A., Montanelli, S.: Matching ontologies in open networked systems: techniques and applications. *J. Data Semant.* **V**, 25–63 (2006) (p. 207)
- Cerbah, F., Euzenat, J.: Traceability between models and texts through terminology. *Data Knowl. Eng.* **38**(1), 31–43 (2001) (p. 98)
- Chai, X., Sayyadian, M., Doan, A., Rosenthal, A., Seligman, L.: Analyzing and revising mediated schemas to improve their matchability. *Proc. VLDB Endow.* **1**(1), 773–784 (2008) (p. 265)
- Chalupsky, H.: OntoMorph: a translation system for symbolic knowledge. In: Proc. 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR), Breckenridge, CO, USA, pp. 471–482 (2000) (pp. 38, 39)
- Chan, L.M., Comaromi, J., Mitchell, J., Satija, M.: Dewey Decimal Classification: a Practical Guide. OCLC Forest Press, Dublin (1996) (p. 28)
- Chang, K.C.-C.: Deep-web search. In: Liu, L., Özsu, T. (eds.) *Encyclopedia of Database Systems*, pp. 784–788. Springer, Berlin (2009) (pp. 21, 23)
- Chang, K., He, B., Li, C., Patel, M., Zhang, Z.: Structured databases on the web: observations and implications. *SIGMOD Rec.* **33**(3), 61–70 (2004) (p. 23)
- Chang, K., He, B., Zhang, Z.: Toward large scale integration: building a metaquerier over databases on the web. In: Proc. 2nd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, pp. 44–55 (2005) (p. 221)
- Chawathe, S., Garcia Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS project: integration of heterogeneous information sources. In: Proc. 16th Meeting of the Information Processing Society of Japan (IPSJ), Tokyo, Japan, pp. 7–18 (1994) (pp. 5, 201)
- Chen, P.: The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.* **1**(1), 9–36 (1976). doi:[10.1145/320434.320440](https://doi.org/10.1145/320434.320440) (p. 31)
- Chen, K., Kannan, A., Madhavan, J., Halevy, A.: Exploring schema repositories with Schemr. *SIGMOD Rec.* **40**(1), 11–16 (2011) (pp. 5, 349)

- Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S. (eds.): Web Services Description Language (WSDL) version 2.0—Part 1: Core language. Recommendation, W3C (2007) (p. 31)
- Choi, N., Song, I.-Y., Han, H.: A survey on ontology mapping. *SIGMOD Rec.* **35**(3), 34–41 (2006) (pp. 73, 201)
- Clark, J., DeRose, S. (eds.): XML path language (XPath) version 1.0. Recommendation, W3C (1999) (p. 324)
- Clifton, C., Hausman, E., Rosenthal, A.: Experience with a combined approach to attribute matching across heterogeneous databases. In: Proc. 7th IFIP Conference on Database Semantics, Leysin, Switzerland, pp. 428–453 (1997) (pp. 58, 203)
- Cohen, W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. In: Proc. 17th International Conference on Management of Data (SIGMOD), Seattle, WA, USA, pp. 201–212 (1998) (p. 174)
- Cohen, W., Hirsh, H.: Joins that generalize: text classification using WHIRL. In: Proc. 4th International Conference on Knowledge Discovery and Data Mining (KDD), New York, NY, USA, pp. 169–173 (1998) (p. 174)
- Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: Proc. International Workshop on Information Integration on the Web (IIWeb) at the 18th International Joint Conference on Artificial Intelligence IJCAI, Acapulco, Mexico, pp. 73–78 (2003a) (p. 239)
- Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string metrics for matching names and records. In: Proc. International Workshop on Data Cleaning and Object Consolidation at the 9th International Conference on Knowledge Discovery and Data Mining (KDD), Washington, DC, USA (2003b) (pp. 88, 96)
- Conroy, C., Brennan, R., O’Sullivan, D., Lewis, D.: User evaluation study of a tagging approach to semantic mapping. In: Proc. 6th European Semantic Web Conference (ESWC), Heraklion, Greece, Lecture Notes in Computer Science, vol. 5554, pp. 623–637 (2009) (p. 357)
- Corcho, Ó.: A Layered Declarative Approach to Ontology Translation with Knowledge Preservation. IOS Press, Amsterdam (2005) (pp. 37, 55)
- Corcho, Ó., Gómez-Pérez, A.: ODEDialect: a set of declarative languages for implementing ontology translation systems. *J. Univers. Comput. Sci.* **13**(12), 1805–1834 (2007) (p. 322)
- Corrales, J.C., Grigori, D., Bouzeghoub, M., Burbano, J.E.: BeMatch: a platform for matchmaking service behavior models. In: Proc. 11th International Conference on Extending Database Technology (EDBT), Nantes, France, pp. 695–699 (2008) (p. 227)
- Correa, E.S., Freitas, A.A., Johnson, C.: A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In: Proc. 8th Genetic and Evolutionary Computation Conference (GECCO), Seattle, WA, USA, pp. 35–42 (2006) (p. 139)
- Correndo, G., Alani, H., Smart, P.: A community based approach for managing ontology alignments. In: Proc. 3rd International Workshop on Ontology Matching (OM) at the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany, pp. 61–72 (2008) (p. 358)
- Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N.: SPARQL query rewriting for implementing data integration over linked data. In: Proc. 1st International Workshop on Data Semantics (DataSem) at the 13th International Conference on Extending Database Technology (EDBT), Lausanne, Switzerland (2010) (p. 389)
- Cortes, C., Vapnik, V.: Support-vector networks. In: Proc. 12th International Conference on Machine Learning (ICML), Tahoe City, CA, USA, pp. 273–297 (1995) (p. 177)
- Coutaz, J., Crowley, J., Dobson, S., Garlan, D.: Context is key. *Commun. ACM* **48**(3), 49–53 (2005) (p. 20)
- Cramer, E.: Probability measures with given marginals and conditionals: *i*-projections and conditional iterative proportional fitting. *Stat. Decis.* **18**(3), 311–329 (2000) (p. 220)
- Cristianini, N., Shawe-Taylor, J.: Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge (2000) (p. 177)
- Cruz, I., Sunna, W.: Structural alignment methods with applications to geospatial ontologies. *Trans. GIS* **12**(6), 683–711 (2008) (p. 230)

- Cruz, I., Antonelli, F.P., Stroe, C.: AgreementMaker: efficient matching for large real-world schemas and ontologies. *Proc. VLDB Endow.* **2**(2), 1586–1589 (2009) (p. 230)
- Cuenca Grau, B., Parsia, B., Sirin, E.: Combining OWL ontologies using \mathcal{E} -connections. *J. Web Semant.* **4**(1), 40–59 (2006) (p. 50)
- Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M., Saggion, H., Petrak, J., Li, Y., Peters, W.: Text Processing with GATE (Version 6). University of Sheffield, Sheffield (2011) (p. 97)
- Damerau, F.: A technique for computer detection and selection of spelling errors. *Commun. ACM* **7**(3), 171–176 (1964) (p. 91)
- d'Aquin, M.: Formally measuring agreement and disagreement in ontologies. In: Proc. 5th International Conference on Knowledge Capture (K-CAP), Redondo Beach, CA, USA, pp. 145–152 (2009) (p. 193)
- d'Aquin, M., Lewen, H.: Cupboard—a place to expose your ontologies to applications and the community. In: Proc. 6th European Semantic Web Conference (ESWC), Demo Track, Herounissos, Greece. Lecture Notes in Computer Science, vol. 5554, pp. 913–918 (2009) (p. 391)
- d'Aquin, M., Motta, E.: Watson, more than a semantic web search engine. *Semant. Web* **2**(1), 55–63 (2011) (pp. 59, 224)
- David, J., Euzenat, J.: Comparison between ontology distances (preliminary results). In: Proc. 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany. Lecture Notes in Computer Science, vol. 5318, pp. 245–260 (2008) (p. 157)
- David, J., Euzenat, J.: On fixing semantic alignment evaluation measures. In: Proc. 3rd International Workshop on Ontology Matching (OM) at the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany, pp. 25–36 (2008) (p. 309)
- David, J., Guillet, F., Briand, H.: Matching directories and OWL ontologies with AROMA. In: Proc. 15th ACM Conference on Information and Knowledge Management (CIKM), Arlington, VA, USA, pp. 830–831 (2006) (p. 252)
- David, J., Guillet, F., Briand, H.: Association rule ontology matching approach. *Int. J. Semantic Web Inf. Syst.* **3**(2), 27–49 (2007) (pp. 152, 252)
- David, J., Euzenat, J., Šváb-Zamazal, O.: Ontology similarity in the alignment space. In: Proc. 9th International Semantic Web Conference (ISWC), Shanghai, China. Lecture Notes in Computer Science, vol. 6496, pp. 129–144 (2010) (p. 157)
- David, J., Euzenat, J., Scharffe, F., Trojahn, C.: The Alignment API 4.0. *Semant. Web* **2**(1), 3–10 (2011) (p. 346)
- Davis, M., Putnam, H.: A computing procedure for quantification theory. *J. ACM* **7**(3), 201–215 (1960) (pp. 339, 367)
- Davis, M., Longemann, G., Loveland, D.: A machine program for theorem proving. *Commun. ACM* **5**(7), 394–397 (1962) (pp. 339, 367)
- de Brujin, J., Foxvog, D., Zimmerman, K.: Ontology mediation patterns library. Deliverable D4.3.1, SEKT (2004) (p. 331)
- Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proc. 6th Symposium on Operating System Design and Implementation (OSDI), San Francisco, CA, USA, pp. 137–150 (2004) (p. 151)
- Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.* **41**(6), 391–407 (1990) (p. 94)
- Dempster, A.: Upper and lower probabilities induced by a multivalued mapping. *Ann. Math. Stat.* **38**(2), 325–339 (1967) (p. 167)
- Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. Ser. B Stat. Methodol.* **39**(1), 1–38 (1977) (pp. 137, 256)
- Dessloch, S., Hernández, M., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: integrating schema mapping and ETL. In: Proc. 24th International Conference on Data Engineering (ICDE), Cancún, Mexico, pp. 1307–1316 (2008) (pp. 5, 384)
- Dewaraja, S.: CLONTY: a scalable approach to collaborative ontology alignment. B.Sc. thesis, The University of Westminster, Westminster, UK (2010) (p. 358)

- Dhamankar, R., Lee, Y., Doan, A.-H., Halevy, A., Domingos, P.: iMAP: discovering complex semantic matches between database schemas. In: Proc. 23rd International Conference on Management of Data (SIGMOD), Paris, France, pp. 383–394 (2004) (pp. 236, 360, 362, 364, 367)
- Dieng, R., Hug, S.: Comparison of “personal ontologies” represented through conceptual graphs. In: Proc. 13th European Conference on Artificial Intelligence (ECAI), Brighton, UK, pp. 341–345 (1998) (p. 126)
- Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., Kolari, P.: Finding and ranking knowledge on the semantic web. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 156–170 (2005) (p. 224)
- Do, H.-H.: Schema matching and mapping-based data integration. PhD thesis, University of Leipzig, Leipzig, Germany (2005) (pp. 83, 342, 343)
- Do, H.-H., Rahm, E.: COMA—a system for flexible combination of schema matching approaches. In: Proc. 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, pp. 610–621 (2002) (pp. 128, 188, 210, 342)
- Do, H.-H., Rahm, E.: Matching large schemas: approaches and evaluation. *Inf. Syst.* **32**(6), 857–885 (2007) (p. 181)
- Do, H.-H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Proc. Workshop on Web, Web-Services, and Database Systems, Erfurt, Germany. Lecture Notes in Computer Science, vol. 2593, pp. 221–237 (2002) (pp. 201, 292, 301)
- Doan, A.-H., Halevy, A.: Semantic integration research in the database community: a brief survey. *AI Mag.* **26**(1), 83–94 (2005). Special issue on Semantic integration (pp. 83, 201)
- Doan, A.-H., Domingos, P., Halevy, A.: Reconciling schemas of disparate data sources: a machine-learning approach. In: Proc. 20th International Conference on Management of Data (SIGMOD), Santa Barbara, CA, USA, pp. 509–520 (2001) (p. 235)
- Doan, A.-H., Domingos, P., Halevy, A.: Learning to match the schemas of data sources: a multi-strategy approach. *Mach. Learn.* **50**(3), 279–301 (2003) (pp. 173, 174, 182, 183)
- Doan, A.-H., Madhavan, J., Domingos, P., Halevy, A.: Ontology matching: a machine learning approach. In: Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, pp. 385–404. Springer, Berlin (2004). Chap. 18 (pp. 173, 235)
- Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann, San Mateo (2012). 497 pp. (pp. 5, 73)
- Domingos, P., Pazzani, M.: Beyond independence: conditions for the optimality of the simple Bayesian classifier. In: Proc. 13th International Conference on Machine Learning (ICML), Bari, Italy, pp. 105–112 (1996) (p. 173)
- Doran, P., Tamma, V., Payne, T., Palmisano, I.: Dynamic selection of ontological alignments: a space reduction mechanism. In: Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, CA, USA, pp. 2028–2033 (2009) (p. 151)
- Doshi, P., Kolli, R., Thomas, C.: Inexact matching of ontology graphs using expectation-maximization. *J. Web Semant.* **7**(2), 90–106 (2009) (pp. 138, 256)
- Dou, D., McDermott, D., Qi, P.: Ontology translation on the semantic web. *J. Data Semant.* **II**, 35–57 (2005) (p. 380)
- Dowling, W., Gallier, J.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Log. Program.* **1**(3), 267–284 (1984) (p. 261)
- Dragut, E., Lawrence, R.: Composing mappings between schemas using a reference ontology. In: Proc. 3rd International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Larnaca, Cyprus. Lecture Notes in Computer Science, vol. 3290, pp. 783–800 (2004) (p. 343)
- Draper, D., Halevy, A., Weld, D.: The nimble integration engine. In: Proc. 20th International Conference on Management of Data SIGMOD, Santa Barbara, CA, USA, pp. 567–568 (2001) (p. 5)
- Drumm, C., Schmitt, M., Do, H.-H., Rahm, E.: QuickMig: automatic schema matching for data migration projects. In: Proc. 16th ACM Conference on Information and Knowledge Management (CIKM), Lisbon, Portugal, pp. 107–116 (2007) (p. 211)
- Duchateau, F., Bellahsene, Z., Hunt, E.: X BenchMatch: a benchmark for XML schema matching tools. In: Proc. 33rd Very Large Databases (VLDB) Demo Track, Wien, Austria, pp. 1318–1321

- (2007a) (p. 299)
- Duchateau, F., Bellahsene, Z., Roche, M.: A context-based measure for discovering approximate semantic matching between schema elements. In: Proc. 1st International Conference on Research in Information Science (RCIS), Ouarzazate, Morocco, pp. 9–20 (2007b) (p. 228)
- Duchateau, F., Bellahsene, Z., Coletta, R.: A flexible approach for planning schema matching algorithms. In: Proc. 16th International Conference on Cooperative Information Systems (CoopIS), Monterrey, Mexico. Lecture Notes in Computer Science, vol. 5331, pp. 249–264 (2008) (pp. 178, 181, 228)
- Duchateau, F., Coletta, R., Bellahsene, Z., Miller, R.: (not) Yet Another Matcher. In: Proc. 18th ACM Conference on Information and Knowledge Management (CIKM), Hong Kong, China, pp. 1537–1540 (2009) (pp. 178, 258)
- Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n -person games. *Artif. Intell.* **77**(2), 321–358 (1995) (p. 171)
- Dzbor, M., Domingue, J., Motta, E.: Magpie—towards a semantic web browser. In: Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, USA. Lecture Notes in Computer Science, vol. 2870, pp. 690–705 (2003) (p. 21)
- Dzbor, M., Motta, E., Domingue, J.: Opening up Magpie via semantic services. In: Proc. 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan. Lecture Notes in Computer Science, vol. 3298, pp. 635–649 (2004) (p. 21)
- Eckert, K., Meilicke, C., Stuckenschmidt, H.: Improving ontology matching using meta-level learning. In: Proc. 6th European Semantic Web Conference (ESWC), Hersounisous, Greece. Lecture Notes in Computer Science, vol. 5554, pp. 158–172 (2009) (p. 181)
- Ehrig, M.: Ontology Alignment: Bridging the Semantic Gap. Springer, New York (2007). 0-387-32805-X (pp. 82, 123, 263, 314, 348)
- Ehrig, M., Euzenat, J.: Relaxed precision and recall for ontology matching. In: Proc. International Workshop on Integrating Ontologies at the 3rd International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 25–32 (2005) (pp. 307, 308)
- Ehrig, M., Staab, S.: QOM—quick ontology mapping. In: Proc. 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan. Lecture Notes in Computer Science, vol. 3298, pp. 683–697 (2004) (pp. 244, 312)
- Ehrig, M., Sure, Y.: Ontology mapping—an integrated approach. In: Proc. 1st European Semantic Web Symposium (ESWS), Hersounisous, Greece. Lecture Notes in Computer Science, vol. 3053, pp. 76–91 (2004) (pp. 74, 112, 126, 188, 189, 244)
- Ehrig, M., Staab, S., Sure, Y.: Bootstrapping ontology alignment methods with APFEL. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 186–200 (2005) (pp. 175, 177, 178, 262)
- El Jerroudi, Z., Ziegler, J.: iMERGE: interactive ontology merging. In: Proc. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Demo Track, Acitrezza, Italy, pp. 52–56 (2008) (p. 371)
- Elfeky, M., Elmagarmid, A., Verykios, V.: TAILOR: a record linkage tool box. In: Proc. 18th International Conference on Data Engineering (ICDE), San Jose, CA, USA, pp. 17–28 (2002) (p. 117)
- Elmagarmid, A., Rusinkiewicz, M., Sheth, A. (eds.): Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann, San Francisco (1999) (p. 8)
- Elmeleegy, H., Ouzzani, M., Elmagarmid, A.: Usage-based schema matching. In: Proc. 24th International Conference on Data Engineering (ICDE), Cancún, Mexico, pp. 20–29 (2008) (pp. 185, 226)
- Embley, D., Xu, L., Ding, Y.: Automatic direct and indirect schema mapping: experiences and lessons learned. *SIGMOD Rec.* **33**(4), 14–19 (2004) (p. 245)
- Engelbrecht, A.P.: Fundamentals of Computational Swarm Intelligence. Wiley, New York (2005) (p. 139)
- Esposito, F., Fanizzi, N., d'Amato, C.: Recovering uncertain mappings through structural validation and aggregation with the MoTo system. In: Proc. 25th ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, pp. 1428–1432 (2010) (pp. 173, 175, 182, 259)

- Euzenat, J.: Brief overview of T-tree: the Tropes Taxonomy building Tool. In: Proc. 4th ASIS SIG/CR Workshop on Classification Research, Columbus, OH, USA, pp. 69–87 (1994) (p. 233)
- Euzenat, J.: Towards a principled approach to semantic interoperability. In: Proc. International Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, USA, pp. 19–25 (2001) (pp. 37, 38, 39)
- Euzenat, J.: Towards composing and benchmarking ontology alignments. In: Proc. International Workshop on Semantic Integration at the 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, USA, pp. 165–166 (2003) (pp. 47, 289)
- Euzenat, J.: An API for ontology alignment. In: Proc. 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan. Lecture Notes in Computer Science, vol. 3298, pp. 698–712 (2004) (pp. 42, 329)
- Euzenat, J.: Alignment infrastructure for ontology mediation and other applications. In: Proc. International Workshop on Mediation in Semantic Web Services (MEDIATE) at the 3rd International Conference on Service Oriented Computing (ICSOC), Amsterdam, The Netherlands, pp. 81–95 (2005) (pp. 340, 391)
- Euzenat, J.: Semantic precision and recall for ontology alignment evaluation. In: Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, pp. 348–353 (2007) (pp. 52, 309)
- Euzenat, J.: Algebras of ontology alignment relations. In: Proc. 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany. Lecture Notes in Computer Science, vol. 5318, pp. 387–402 (2008) (p. 43)
- Euzenat, J., Le Duc, C.: Methodological guidelines for matching ontologies. In: Suárez Figueroa, M., Gómez Pérez, A., Motta, E., Gangemi, A. (eds.) *Ontology Engineering in a Networked World*, pp. 257–278. Springer, Heidelberg (2012). Chap. 12 (pp. 55, 350)
- Euzenat, J., Stuckenschmidt, H.: The ‘family of languages’ approach to semantic interoperability. In: Omelchenko, B., Klein, M. (eds.) *Knowledge Transformation for the Semantic Web*, pp. 49–63. IOS Press, Amsterdam (2003) (p. 38)
- Euzenat, J., Valtchev, P.: Similarity-based ontology alignment in OWL-lite. In: Proc. 16th European Conference on Artificial Intelligence (ECAI), Valencia, Spain, pp. 333–337 (2004) (pp. 111, 123, 126, 134, 247)
- Euzenat, J., Bach, T.L., Barrasa, J., Bouquet, P., De Bo, J., Dieng-Kuntz, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckenschmidt, H., Shvaiko, P., Tessaris, S., Van Acker, S., Zaihrayeu, I.: State of the art on ontology alignment. Deliverable D2.2.3, Knowledge web NoE (2004) (pp. 98, 122, 123)
- Euzenat, J., Laera, L., Tamma, V., Viollet, A.: Negotiation/argumentation techniques among agents complying to different ontologies. Deliverable 2.3.7, Knowledge web NoE (2005) (p. 19)
- Euzenat, J., Ehrig, M., Jentsch, A., Mochol, M., Shvaiko, P.: Case-based recommendation of matching tools and techniques. Deliverable 1.2.2.2.1, Knowledge web (2006) (p. 61)
- Euzenat, J., Isaac, A., Meilicke, C., Shvaiko, P., Stuckenschmidt, H., Šváb, O., Svátek, V., van Hage, W.R., Yatskevich, M.: Results of the ontology alignment evaluation initiative 2007. In: Proc. 2nd International Workshop on Ontology Matching (OM) at the 6th International Semantic Web Conference (ISWC) and the 2nd Asian Semantic Web Conference (ASWC), Busan, Korea, pp. 96–132 (2007a) (p. 311)
- Euzenat, J., Scharffe, F., Zimmermann, A.: Expressive alignment language and implementation. Deliverable 2.2.10, Knowledge web (2007b) (p. 331)
- Euzenat, J., Mocan, A., Scharffe, F.: Ontology alignments: an ontology management perspective. In: Hepp, M., De Leenheer, P., De Moor, A., Sure, Y. (eds.) *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*, pp. 177–206. Springer, New York (2008a). Chap. 6 (pp. 55, 56, 57, 378, 394, 395)
- Euzenat, J., Pierson, J., Ramparani, F.: Dynamic context management for pervasive applications. *Knowl. Eng. Rev.* **23**(1), 21–49 (2008b) (p. 20)
- Euzenat, J., Scharffe, F., Polleres, A.: Processing ontology alignments with SPARQL (position paper). In: Proc. 1st International Workshop on Ontology Alignment and Visualization (OnAV)

- at the 2nd International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), Barcelona, Spain, pp. 913–917 (2008c) (p. 382)
- Euzenat, J., Meilicke, C., Shvaiko, P., Stuckenschmidt, H., Trojahn, C.: Ontology Alignment Evaluation Initiative: six years of experience. *J. Data Semant.* **XV**, 158–192 (2011) (pp. 288, 289, 298, 317)
- Euzenat, J., Roșoiu, M., Trojahn, C.: Ontology matching benchmarks: generation, stability, and discriminability. *J. Web Semant.* **21**, 30–48 (2013) (p. 299)
- Everitt, B.: Cluster Analysis. Wiley, New York (1993) (p. 213)
- Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, Cambridge (1995) (pp. 49, 52, 53)
- Fagin, R., Kolaitis, P., Popa, L., Tan, W.-C.: Composing schema mappings: second-order dependencies to the rescue. *ACM Trans. Database Syst.* **30**(4), 994–1005 (2005) (p. 384)
- Fagin, R., Haas, L., Hernández, M., Miller, R., Popa, L., Velegrakis, Y.: Clio: schema mapping creation and data exchange. In: Borgida, A., Chaudhri, V., Giorgini, P., Yu, E. (eds.) Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science, vol. 5600, pp. 198–236. Springer, Berlin (2009) (p. 383)
- Falconer, S., Storey, M.-A.: A cognitive support framework for ontology mapping. In: Proc. 6th International Semantic Web Conference (ISWC) and 2nd Asian Semantic Web Conference (ASWC), Busan, Korea. Lecture Notes in Computer Science, vol. 4825, pp. 114–127 (2007) (p. 375)
- Falconer, S., Bull, I., Grammel, L., Storey, M.-A.: Creating visualizations through ontology mapping. In: Proc. 3rd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), Fukuoka, Japan, pp. 688–693 (2009) (p. 375)
- Fellbaum, C.: WordNet: an Electronic Lexical Database. MIT Press, Cambridge (1998) (p. 99)
- Fellegi, I., Sunter, A.: A theory for record linkage. *J. Am. Stat. Assoc.* **64**(328), 1183–1210 (1969) (p. 117)
- Fensel, D.: Ontologies: a Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edn. Springer, Heidelberg (2004) (p. vii)
- Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: the Web Service Modeling Ontology. Springer, Heidelberg (2007) (pp. 16, 17)
- Ferrara, A., Lorusso, D., Montanelli, S., Varese, G.: Towards a benchmark for instance matching. In: Proc. 3rd International Workshop on Ontology Matching (OM) at the 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany, pp. 37–48 (2008) (p. 299)
- Ferrara, A., Montanelli, S., Noessner, J., Stuckenschmidt, H.: Benchmarking matching applications on the semantic web. In: Proc. 8th Extended Semantic Web Conference (ESWC), Heraklion, Greece. Lecture Notes in Computer Science, vol. 6644, pp. 108–122 (2011a) (p. 299)
- Ferrara, A., Nikolov, A., Scharffe, F.: Data linking for the semantic web. *Int. J. Semantic Web Inf. Syst.* **7**(3), 46–76 (2011b) (pp. 115, 385)
- FIPA: FIPA ACL communicative act library specification. Technical report, FIPA (2002a). <http://www.fipa.org/specs/fipa00037> (p. 19)
- FIPA: FIPA ACL message structure specification. Technical report, FIPA (2002b). <http://www.fipa.org/specs/fipa00061> (p. 19)
- Fleischhacker, D., Stuckenschmidt, H.: A practical implementation of semantic precision and recall. In: Proc. 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), Krakow, Poland, pp. 986–991 (2010) (p. 309)
- Fleiss, J.: Statistical Methods for Rates and Proportions. Wiley, Chichester (1973) (p. 238)
- Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: classification and survey. *Knowl. Eng. Rev.* **23**(2), 117–152 (2008) (p. 73)
- Fowler, J., Perry, B., Nodine, M., Bargmeyer, B.: Agent-based semantic interoperability in InfoSleuth. *SIGMOD Rec.* **28**(1), 60–67 (1999) (p. 201)
- Franconi, E., Kuper, G., Lopatenko, A., Serafini, L.: A robust logical and computational characterisation of peer-to-peer database systems. In: Proc. International Workshop on Databases, Infor-

- mation Systems and Peer-to-Peer Computing (DBISP2P) at the 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, pp. 64–76 (2003) (p. 45)
- Fu, B., Brennan, R., O’Sullivan, D.: A configurable translation-based cross-lingual ontology mapping system to adjust mapping outcomes. *J. Web Semant.* **15**, 15–36 (2012) (p. 105)
- Gal, A.: Managing uncertainty in schema matching with top- k schema mappings. *J. Data Semant.* **VI**, 90–114 (2006) (p. 209)
- Gal, A.: Uncertain Schema Matching. Synthesis Lectures on Data Management. Morgan & Claypool, San Rafael (2011) (pp. 44, 164, 181, 267)
- Gal, A., Shvaiko, P.: Advances in ontology matching. In: Dillon, T., Chang, E., Meersman, R., Sycara, K. (eds.) *Advances in Web Semantics i*, pp. 176–198. Springer, Heidelberg (2009) (p. 181)
- Gal, A., Anaby-Tavor, A., Trombetta, A., Montesi, D.: A framework for modeling and evaluating automatic semantic reconciliation. *VLDB J.* **14**(1), 50–67 (2005a) (p. 165)
- Gal, A., Modica, G., Jamil, H., Eyal, A.: Automatic ontology matching using application semantics. *AI Mag.* **26**(1), 21–32 (2005b) (p. 209)
- Gale, D., Shapley, L.S.: College admissions and the stability of marriage. *Am. Math. Mon.* **69**(1), 5–15 (1962) (p. 190)
- Gangemi, A.: Restructuring semi-structured terminologies for ontology building: a realistic case study in fishery information systems. Deliverable D16, WonderWeb (2004) (p. 156)
- Gangemi, A., Guarino, N., Masolo, C., Oltramari, A.: Sweetening WordNet with DOLCE. *AI Mag.* **24**(3), 13–24 (2003) (p. 32)
- Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin (1999) (p. 114)
- García Castro, R., Maynard, D., Foxvog, D., Wache, H., González-Cabero, R.: Specification of a methodology, general criteria, and benchmark suites for benchmarking ontology tools. Deliverable D2.1.4, Knowledge web NoE (2004) (p. 286)
- Gene Ontology Consortium: The Gene Ontology project in 2008. *Nucleic Acids Res.* **36**, D440–D444 (2008) (p. 4)
- Ghazvinian, A., Noy, N., Jonquet, C., Shah, N., Musen, M.: What four million mappings can tell you about two hundred ontologies. In: Proc. 8th International Semantic Web Conference (ISWC), Chantilly, VA, USA. Lecture Notes in Computer Science, vol. 5823, pp. 229–242 (2009) (p. 392)
- Ghazvinian, A., Noy, N., Musen, M.: From mappings to modules: using mappings to identify domain-specific modules in large ontologies. In: Proc. 6th International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 33–40 (2011) (p. 151)
- Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning = locality + compatibility. *Artif. Intell.* **127**(2), 221–259 (2001) (p. 51)
- Ghidini, C., Giunchiglia, F.: A semantics for abstraction. In: Proc. 16th European Conference on Artificial Intelligence (ECAI), Valencia, Spain, pp. 343–347 (2004) (p. 37)
- Ghidini, C., Serafini, L.: Distributed first order logics. In: Proc. 2nd Conference on Frontiers of Combining Systems (FroCoS), Amsterdam, The Netherlands, pp. 121–139 (1998) (p. 45)
- Giunchiglia, F., Shvaiko, P.: Semantic matching. *Knowl. Eng. Rev.* **18**(3), 265–280 (2003) (pp. 75, 145, 213)
- Giunchiglia, F., Yatskevich, M.: Element level semantic matching. In: Proc. International Workshop on Meaning Coordination and Negotiation at the 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan, pp. 37–48 (2004) (p. 104)
- Giunchiglia, F., Zaihrayeu, I.: Making peer databases interact—a vision for an architecture supporting data coordination. In: Proc. 6th International Workshop on Cooperative Information Agents (CIA), Madrid, Spain, pp. 18–35 (2002) (p. 15)
- Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: an algorithm and an implementation of semantic matching. In: Proc. 1st European Semantic Web Symposium (ESWS), Hersounisou, Greece. Lecture Notes in Computer Science, vol. 3053, pp. 61–75 (2004) (pp. 75, 101, 214)
- Giunchiglia, F., Shvaiko, P., Yatskevich, M.: Semantic schema matching. In: Proc. 13rd International Conference on Cooperative Information Systems (CoopIS), Agia Napa, Cyprus. Lecture

- Notes in Computer Science, vol. 3761, pp. 347–365 (2005a) (p. 6)
- Giunchiglia, F., Yatskevich, M., Giunchiglia, E.: Efficient semantic matching. In: Proc. 2nd European Semantic Web Conference (ESWC), Hersounis, Greece. Lecture Notes in Computer Science, vol. 3532, pp. 272–289 (2005b) (p. 215)
- Giunchiglia, F., Marchese, M., Zaihrayeu, I.: Encoding classifications into lightweight ontologies. In: Proc. 3rd European Semantic Web Conference (ESWC), Budva, Montenegro. Lecture Notes in Computer Science, vol. 4011, pp. 80–94 (2006a) (p. 27)
- Giunchiglia, F., McNeill, F., Yatskevich, M.: Web service composition via semantic matching of interaction specifications. Technical Report DIT-06-080, University of Trento (2006b) (p. 17)
- Giunchiglia, F., Shvaiko, P., Yatskevich, M.: Discovering missing background knowledge in ontology matching. In: Proc. 17th European Conference on Artificial Intelligence (ECAI), Riva del Garda, Italy, pp. 382–386 (2006c) (pp. 104, 156)
- Giunchiglia, F., Yatskevich, M., Shvaiko, P.: Semantic matching: algorithms and implementation. *J. Data Semant.* **IX**, 1–38 (2007) (p. 214)
- Giunchiglia, F., Yatskevich, M., McNeill, F., Pane, J., Besana, P., Shvaiko, P.: Approximate structure preserving semantic matching. In: Proc. 7th of International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Monterrey, Mexico. Lecture Notes in Computer Science, vol. 5332, pp. 1217–1234 (2008) (p. 215)
- Giunchiglia, F., Yatskevich, M., Avesani, P., Shvaiko, P.: A large scale dataset for the evaluation of ontology matching systems. *Knowl. Eng. Rev.* **24**(2), 137–157 (2009) (p. 291)
- Giunchiglia, F., Autayeu, A., Pane, J.: S-Match: an open source framework for matching lightweight ontologies. *Semant. Web* **3**(3), 307–317 (2012a) (p. 214)
- Giunchiglia, F., Maltese, V., Autayeu, A.: Computing minimal mappings between lightweight ontologies. *Int. J. Digit. Libr.* **12**(4), 179–193 (2012b) (p. 215)
- Goasdoué, F., Lattes, V., Rousset, M.-C.: The use of CARIN language and algorithms for information integration: the PICSEL system. *Int. J. Coop. Inf. Syst.* **9**(4), 383–401 (2000) (p. 201)
- Goh, C.-H.: Representing and reasoning about semantic conflicts in heterogeneous information sources. PhD thesis, MIT, Cambridge, MA, USA (1997) (p. 37)
- Good, I.J.: The Estimation of Probabilities: an Essay on Modern Bayesian Methods. MIT Press, Cambridge (1965) (p. 173)
- Gotoh, O.: An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162**(3), 705–708 (1981) (p. 92)
- Gracia, J.: Integration and disambiguation techniques for semantic heterogeneity reduction on the web. PhD thesis, Universidad de Zaragoza, Zaragoza, Spain (2009) (p. 99)
- Gracia, J., Lopez, V., d'Aquin, M., Sabou, M., Motta, E., Mena, E.: Solving semantic ambiguity to improve semantic web based ontology matching. In: Proc. 2nd International Workshop on Ontology Matching (OM) at the 6th International Semantic Web Conference (ISWC) and the 2nd Asian Semantic Web Conference (ASWC), Busan, Korea, pp. 1–12 (2007) (p. 224)
- Gracia, J., Bernad, J., Mena, E.: Ontology matching with CIDER: evaluation report for OAEI 2011. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 126–133 (2011) (pp. 175, 226)
- Gridinoc, L., Sabou, M., d'Aquin, M., Dzbor, M., Motta, E.: Semantic browsing with PowerMagpie. In: Proc. 5th European Semantic Web Conference (ESWC), Tenerife, Spain. Lecture Notes in Computer Science, vol. 5021, pp. 802–806 (2008) (p. 21)
- Grigori, D., Corrales, J.C., Bouzeghoub, M.: Behavioral matchmaking for service retrieval: application to conversation protocols. *Inf. Syst.* **33**(7–8), 681–698 (2008) (p. 227)
- Guarino, N.: The ontological level: revisiting 30 years of knowledge representation. In: Borgida, A., Chaudhri, V., Giorgini, P., Yu, E. (eds.) Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science, vol. 5600, pp. 52–67. Springer, Berlin (2009) (p. vii)
- Guha, S., Rastogi, R., Shim, K.: ROCK: a robust clustering algorithm for categorical attributes. In: Proc. 15th International Conference on Data Engineering (ICDE), Sydney, Australia, pp. 512–521 (1999) (p. 151)
- Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, University of Twente (2005) (p. 32)

- Haase, P., Schnizler, B., Broekstra, J., Ehrig, M., van Harmelen, F., Menken, M., Mika, P., Plechawski, M., Pyszlak, P., Siebes, R., Staab, S., Tempich, C.: Bibster—a semantics-based bibliographic peer-to-peer system. *J. Web Semant.* **2**(1), 99–103 (2004) (p. 14)
- Haeri (Hosseini), S., Abolhassani, H., Qazvinian, V., Hariri, B.B.: Coincidence-based scoring of mappings in ontology alignment. *J. Adv. Comput. Intell. Inform.* **11**(7), 803–816 (2007) (p. 222)
- Hájek, P.: The Metamathematics of Fuzzy Logic. Kluwer, Dordrecht (1998) (p. 161)
- Halevy, A., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.: Enterprise information integration: successes, challenges and controversies. In: Proc. 24th International Conference on Management of Data (SIGMOD), Baltimore, MD, USA, pp. 778–787 (2005) (pp. 5, 10)
- Hamdi, F., Reynaud, C., Safar, B.: Pattern-based mapping refinement. In: Proc. 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Lisbon, Portugal. Lecture Notes in Computer Science, vol. 6317, pp. 1–15 (2010a) (pp. 130, 232)
- Hamdi, F., Safar, B., Reynaud, C., Zargayouna, H.: Alignment-based partitioning of large-scale ontologies. In: Guillet, F., Ritschard, G., Zighed, D.A., Briand, H. (eds.) Advances in Knowledge Discovery and Management. Studies in Computational Intelligence, vol. 292, pp. 251–269. Springer, Berlin (2010b) (pp. 75, 151, 232)
- Hameed, A., Preece, A., Sleeman, D.: Ontology reconciliation. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 231–250. Springer, Berlin (2004). Chap. 12 (pp. 37, 40)
- Hammer, J., Stonebraker, M., Topsakal, O.: THALIA: Test Harness for the Assessment of Legacy Information Integration Approaches. In: Proc. 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, pp. 485–486 (2005) (p. 297)
- Hamming, R.: Error detecting and error correcting codes. Technical Report 2. Bell Syst. Tech. J. (1950) (p. 89)
- Hanif, M.S., Aono, M.: An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *J. Web Semant.* **7**(4), 344–356 (2009) (pp. 152, 229)
- Hartmann, J., Palma, R., Sure, Y., Haase, P., Suárez-Figueroa, M.C., Haase, P., Gómez-Pérez, A., Studer, R.: Ontology metadata vocabulary and applications. In: Proc. 4th International Conference on Ontologies, Databases and Applications of Semantics (ODBASE), Agia Napa, Cyprus. Lecture Notes in Computer Science, vol. 3762, pp. 906–915 (2005) (p. 338)
- Hartung, M., Groß, A., Rahm, E.: COOntoDiff: generation of complex evolution mappings for life science ontologies. *J. Biomed. Inform.* **46**(1), 15–32 (2013) (p. 5)
- Hassanzadeh, O., Kementsietsidis, A., Lim, L., Miller, R., Wang, M.: A framework for semantic link discovery over relational data. In: Proc. 18th Conference on Information and Knowledge Management (CIKM), Hong Kong, China, pp. 1027–1036 (2009) (p. 387)
- Hausdorff, F.: Grundzüge der Mengenlehre, p. 476. Verlag Veit, Leipzig (1914) (p. 119)
- He, B., Chang, K.: Statistical schema matching across web query interfaces. In: Proc. 22nd International Conference on Management of Data (SIGMOD), San Diego, CA, USA, pp. 217–228 (2003) (p. 240)
- He, B., Chang, K.: Automatic complex schema matching across web query interfaces: a correlation mining approach. *ACM Trans. Database Syst.* **31**(1), 1–45 (2006) (pp. 221, 293)
- He, H., Meng, W., Yu, C., Wu, Z.: Automatic integration of web search interfaces with WISE-integrator. *VLDB J.* **13**(3), 256–273 (2004) (p. 245)
- He, H., Meng, W., Yu, C., Wu, Z.: WISE-integrator: a system for extracting and integrating complex web search interfaces of the deep web. In: Proc. 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway, pp. 1314–1317 (2005) (p. 245)
- He, B., Patel, M., Zhang, Z., Chang, K.C.-C.: Accessing the deep web. *Commun. ACM* **50**(5), 94–101 (2007) (p. 23)
- Heath, T., Bizer, C.: Linked Data: Evolving the Web Into a Global Data Space. Morgan & Claypool, San Rafael (2011) (p. 11)
- Hitzler, P., Euzenat, J., Krötzsch, M., Serafini, L., Stuckenschmidt, H., Wache, H., Zimmermann, A.: Integrated view and comparison of alignment semantics. Deliverable 2.2.5, Knowledge web NoE (2005) (p. 49)

- Holland, J.: *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge (1992) (p. 184)
- Horridge, M., Bail, S., Parsia, B., Sattler, U.: The cognitive complexity of OWL justifications. In: Proc. 10th International Semantic Web Conference (ISWC), Bonn, Germany. Lecture Notes in Computer Science, vol. 7031, pp. 241–256 (2011) (p. 360)
- Horridge, M., Parsia, B., Sattler, U.: Justification masking in ontologies. In: Proc. 13th International Conference on Principles of Knowledge Representation and Reasoning (KR), Rome, Italy, pp. 623–627 (2012) (p. 360)
- Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: a semantic web rule language combining OWL and RuleML (2004). <http://www.w3.org/Submission/SWRL/> (p. 327)
- Hovy, E.: Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In: Proc. 1st International Conference on Language Resources and Evaluation (LREC), Granada, Spain, pp. 535–542 (1998) (p. 203)
- Hu, W., Jian, N., Qu, Y., Wang, Q.: GMO: a Graph Matching for Ontologies. In: Proc. International Workshop on Integrating Ontologies at the 3rd International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 43–50 (2005) (p. 248)
- Hu, W., Qu, Y., Cheng, G.: Matching large ontologies: a divide-and-conquer approach. Data Knowl. Eng. **67**(1), 140–160 (2008) (p. 248)
- Huber, J., Szytyler, T., Noessner, J., Meilicke, C.: CODI: Combinatorial Optimization for Data Integration: results for OAEI 2011. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 134–141 (2011) (p. 260)
- Hull, R.: Managing semantic heterogeneity in databases: a theoretical perspective. In: Proc. 16th Symposium on Principles of Database Systems (PODS), Tucson, AZ, USA, pp. 51–61 (1997) (p. 37)
- Huza, M., Harzallah, M., Trichet, F.: OntoMas: a tutoring system dedicated to ontology matching. In: Proc. 1st International Workshop on Ontology Matching (OM) at the 5th International Semantic Web Conference (ISWC), Athens, GA, USA, pp. 228–323 (2006) (p. 62)
- Ichise, R., Takeda, H., Honiden, S.: Integrating multiple Internet directories by instance-based learning. In: Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, pp. 22–30 (2003) (pp. 6, 237)
- Ichise, R., Hamasaki, M., Takeda, H.: Discovering relationships among catalogs. In: Proc. 7th International Conference on Discovery Science, Padova, Italy. Lecture Notes in Computer Science, vol. 3245, pp. 371–379 (2004) (p. 237)
- Ide, N., Véronis, J.: Word Sense Disambiguation: the state of the art. Comput. Linguist. **24**(1), 1–40 (1998) (p. 99)
- Isaac, A., Matthezing, H., van der Meij, L., Schlobach, S., Wang, S., Zinn, C.: Putting ontology alignment in context: usage scenarios, deployment and evaluation in a library case. In: Proc. 5th European Semantic Web Conference (ESWC), Tenerife, Spain. Lecture Notes in Computer Science, vol. 5021, pp. 402–417 (2008) (p. 316)
- Isaac, A., Wang, S., Zinn, C., Matthezing, H., van der Meij, L., Schlobach, S.: Evaluating thesaurus alignments for semantic interoperability in the library domain. IEEE Intell. Syst. **24**(2), 76–86 (2009) (p. 316)
- Isele, R., Bizer, C.: Active learning of expressive linkage rules using genetic programming. J. Web Semant. (2013, in press) (p. 117)
- Ives, Z., Halevy, A., Mork, P., Tatarinov, I.: Piazza: mediation and integration infrastructure for semantic web data. J. Web Semant. **1**(2), 155–175 (2004) (p. 15)
- Jaccard, P.: Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. Bull. Soc. Vaud. Sci. Nat. **37**, 241–272 (1901) (p. 113)
- Jacquemin, C., Tzoukermann, E.: NLP for term variant extraction: synergy between morphology, lexicon and syntax. In: Strzalkowski, T. (ed.) *Language Information Retrieval*, pp. 25–74. Kluwer, Boston (1999) (p. 98)
- Jain, P., Hitzler, P., Sheth, A., Verma, K., Yeh, P.: Ontology alignment for linked open data. In: Proc. 9th International Semantic Web Conference (ISWC), Shanghai, China. Lecture Notes in

- Computer Science, vol. 6496, pp. 401–416 (2010) (p. 225)
- Jain, P., Yeh, P., Verma, K., Vasquez, R., Damova, M., Hitzler, P., Sheth, A.: Contextual ontology alignment of LOD with an upper ontology: a case study with Proton. In: Proc. 8th Extended Semantic Web Conference (ESWC), Hersounisous, Greece. Lecture Notes in Computer Science, vol. 6643, pp. 80–92 (2011) (p. 226)
- James, N., Todorov, K., Hudelot, C.: Combining visual and textual modalities for multimedia ontology matching. In: Proc. 5th International Conference on Semantic and Digital Media Technologies (SAMT), Saarbrücken, Germany. Lecture Notes in Computer Science, vol. 6725, pp. 95–110 (2010) (p. 241)
- Jaro, M.: UNIMATCH: A record linkage system: User's manual. Technical report, U.S. Bureau of the Census, Washington, DC, USA (1976) (p. 92)
- Jaro, M.: Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J. Am. Stat. Assoc.* **84**(406), 414–420 (1989) (p. 92)
- Jaroszewicz, S., Ivantsynova, L., Scheffer, T.: Schema matching on streams with accuracy guarantees. *Intell. Data Anal.* **12**(3), 253–270 (2008) (p. 241)
- Jean-Mary, Y., Shironoshita, P., Kabuka, M.: Ontology matching with semantic verification. *J. Web Semant.* **7**(3), 235–251 (2009) (p. 254)
- Jeffrey, R.: *The Logic of Decisions*. University of Chicago Press, Chicago (1983) (p. 220)
- Ji, Q., Liu, W., Qi, G., Bell, D.: LCS: A Linguistic Combination System for ontology matching. In: Proc. 1st International Conference on Knowledge Science, Engineering and Management (KSEM), Guilin, China, pp. 176–189 (2006) (p. 263)
- Jian, N., Hu, W., Cheng, G., Qu, Y.: Falcon-AO: aligning ontologies with Falcon. In: Proc. International Workshop on Integrating Ontologies at the 3rd International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 87–93 (2005) (p. 249)
- Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In: Proc. 10th International Conference on Research in Computational Linguistics (ROCLING), Taipei, Taiwan, pp. 19–33 (1997) (pp. 103, 106)
- Jiménez Ruiz, E., Cuenca Grau, B., Horrocks, I., Berlanga, R.: Ontology integration using mappings: towards getting the right logical consequences. In: Proc. 6th European Semantic Web Conference (ESWC), Hersounisous, Greece. Lecture Notes in Computer Science, vol. 5554, pp. 173–188 (2009) (p. 267)
- Jiménez Ruiz, E., Cuenca Grau, B.: LogMap: logic-based and scalable ontology matching. In: Proc. 10th International Semantic Web Conference (ISWC), Bonn, Germany. Lecture Notes in Computer Science, vol. 7031, pp. 273–288 (2011) (p. 260)
- Jiménez Ruiz, E., Cuenca Grau, B., Zhou, Y., Horrocks, I.: Large-scale interactive ontology matching: algorithms and implementation. In: Proc. 20th European Conference on Artificial Intelligence (ECAI), Montpellier, France, pp. 444–449 (2012) (p. 261)
- Jung, J., Euzenat, J.: Towards semantic social networks. In: Proc. 4th European Semantic Web Conference (ESWC), Innsbruck, Austria. Lecture Notes in Computer Science, vol. 4519, pp. 267–280 (2007) (p. 16)
- Jung, J., Häkansson, A., Hartung, R.: Indirect alignment between multilingual ontologies: a case study of Korean and Swedish ontologies. In: Proc. 3rd Symposium on Agents and Multi-agent Systems: Technologies and Applications (KES-AMSTA), Uppsala, Sweden. Lecture Notes in Computer Science, vol. 5559, pp. 233–241 (2009) (p. 105)
- Kalfoglou, Y., Schorlemmer, M.: IF-Map: an ontology mapping method based on information flow theory. *J. Data Semant.* **I**, 98–127 (2003a) (p. 243)
- Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. *Knowl. Eng. Rev.* **18**(1), 1–31 (2003b) (pp. 39, 41, 49, 73, 201)
- Kalfoglou, Y., Schorlemmer, M., Atencia, M.: A formal foundation for ontology-alignment interaction models. *Int. J. Semantic Web Inf. Syst.* **3**(2), 50–68 (2007) (p. 49)
- Kang, J., Naughton, J.: On schema matching with opaque column names and data values. In: Proc. 22nd International Conference on Management of Data (SIGMOD), San Diego, CA, USA, pp. 205–216 (2003) (pp. 78, 238)

- Kashyap, V., Sheth, A.: Semantic and schematic similarities between database objects: a context-based approach. *VLDB J.* **5**(4), 276–304 (1996) (p. 73)
- Kashyap, V., Sheth, A.: Semantic heterogeneity in global information systems: the role of metadata, context and ontologies. In: Papazoglou, M., Schlageter, G. (eds.) *Cooperative Information Systems*, pp. 139–178. Academic Press, New York (1998) (p. 37)
- Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 (1995) (p. 139)
- Kennedy, J., Eberhart, R.: A discrete binary version of the particle swarm algorithm. In: Proc. IEEE International Conference on Systems, Man, and Cybernetics, Orlando, FL, USA, pp. 4104–4108 (1997) (p. 139)
- Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: GeRoMe: a Generic Role-based Metamodel for model management. *J. Data Semant.* **VIII**, 82–117 (2007a) (p. 222)
- Kensche, D., Quix, C., Li, X., Li, Y.: GeRoMeSuite: a system for holistic generic model management. In: Proc. 33rd International Conference on Very Large Data Bases (VLDB), Vienna, Austria, pp. 1322–1325 (2007b) (p. 222)
- Kensche, D., Quix, C., Li, X., Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. *Data Knowl. Eng.* **68**(7), 599–621 (2009) (p. 222)
- Kerrigan, M., Mocan, A., Tanler, M., Fensel, D.: The web service modeling toolkit: an integrated development environment for semantic web services. In: Proc. 4th European Semantic Web Conference (ESWC) System Description Track, Innsbruck, Austria, pp. 303–317 (2007) (p. 370)
- Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. *Computer* **24**(12), 12–18 (1991) (p. 37)
- Kim, J., Jang, M., Ha, Y.-G., Sohn, J.-C., Lee, S.-J.: MoA: OWL ontology merging and alignment tool for the semantic web. In: Proc. 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE), Bari, Italy. Lecture Notes in Computer Science, vol. 3533, pp. 722–731 (2005) (p. 216)
- Kirsten, T., Groß, A., Hartung, M., Rahm, E.: GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *J. Biomed. Semant.* **2**(6), 1–24 (2011) (pp. 5, 344)
- Klein, M.: Combining and relating ontologies: an analysis of problems and solutions. In: Proc. International Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, USA (2001) (pp. 37, 38, 39)
- Kohonen, T.: *Self-Organizing Maps*. Springer, Berlin (2001) (p. 175)
- Köpcke, H., Rahm, E.: Frameworks for entity matching: a comparison. *Data Knowl. Eng.* **69**(2), 197–210 (2010) (p. 115)
- Kotis, K., Vouros, G.: HCONE approach to ontology merging. In: Proc. 1st European Semantic Web Symposium (ESWS), Hersounisous, Greece. Lecture Notes in Computer Science, vol. 3053, pp. 137–151 (2004) (p. 215)
- Kotis, K., Vouros, G., Stergiou, K.: Towards automatic merging of domain ontologies: the HCONE-merge approach. *J. Web Semant.* **4**(1), 60–79 (2006) (p. 215)
- Kschischang, F., Frey, B., Loeliger, H.-A.: Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory* **47**(2), 498–519 (2001) (p. 233)
- Kullback, S., Leibler, R.: On information and sufficiency. *Ann. Math. Stat.* **22**(79), 498–519 (1951) (p. 94)
- Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. *Artif. Intell.* **156**(1), 1–73 (2004) (p. 50)
- Lacher, M., Groh, G.: Facilitating the exchange of explicit knowledge through ontology mappings. In: Proc. 14th International Florida Artificial Intelligence Research Society Conference (FLAIRS), Key West, FL, USA, pp. 305–309 (2001) (p. 234)
- Laera, L., Tamma, V., Euzenat, J., Bench-Capon, T., Payne, T.: Reaching agreement over ontology alignments. In: Proc. 5th International Semantic Web Conference (ISWC), Athens, GA, USA. Lecture Notes in Computer Science, vol. 4273, pp. 371–384 (2006) (pp. 19, 170, 171, 362)

- Laera, L., Blacoe, I., Tamia, V., Payne, T., Euzenat, J., Bench-Capon, T.: Argumentation over ontology correspondences in MAS. In: Proc. 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Honolulu, HI, USA, pp. 1285–1292 (2007) (p. 340)
- Lambrix, P., Kaliyaperumal, R.: A session-based approach for aligning large ontologies. In: Proc. 10th European Semantic Web Conference (ESWC), Montpellier, France. Lecture Notes in Computer Science, vol. 7881, pp. 46–60 (2013) (p. 356)
- Lambrix, P., Liu, Q.: Using partial reference alignments to align ontologies. In: Proc. 6th European Semantic Web Conference (ESWC), Heraklion, Greece. Lecture Notes in Computer Science, vol. 5554, pp. 188–202 (2009) (p. 66)
- Lambrix, P., Tan, H.: SAMBO—a system for aligning and merging biomedical ontologies. *J. Web Semant.* **4**(1), 196–206 (2006) (pp. 173, 252)
- Lambrix, P., Tan, H.: A tool for evaluating ontology alignment strategies. *J. Data Semant.* **VIII**, 182–202 (2007) (p. 252)
- Langlais, P., Véronis, J., Simard, M.: Methods and practical issues in evaluating alignment techniques. In: Proc. 17th International Conference on Computational Linguistics (CoLing), Montréal, Canada, pp. 711–717 (1998) (p. 307)
- Lanzenberger, M., Sampson, J.: AlViz: a tool for visual ontology alignment. In: Proc. 10th International Conference on Information Visualisation (IV), London, UK, pp. 430–440 (2006) (pp. 373, 374)
- Larson, J., Navathe, S., Elmasri, R.: A theory of attributed equivalence in databases with application to schema integration. *IEEE Trans. Softw. Eng.* **15**(4), 449–463 (1989) (pp. 73, 113)
- Le Berre, D., Parrain, A.: The sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.* **7**(2–3), 59–64 (2010) (pp. 215, 367)
- Le Duc, C., Lamolle, M., Zimmermann, A.: An API for distributed reasoning on networked ontologies with alignments. In: Proc. 2nd International Conference on Knowledge Engineering and Ontology Development (KEOD), Valencia, Spain, pp. 295–304 (2010) (p. 390)
- Leacock, C., Chodorow, M., Miller, G.: Using corpus statistics and WordNet relations for sense identification. *Comput. Linguist.* **24**(1), 1–40 (1998) (p. 125)
- Lee, M.L., Yang, L.H., Hsu, W., Yang, X.: XClust: clustering XML schemas for effective integration. In: Proc. 11th International Conference on Information and Knowledge Management (CIKM), McLean, VA, USA, pp. 292–299 (2002) (pp. 111, 212)
- Lee, Y., Sayyadian, M., Doan, A.-H., Rosenthal, A.: eTuner: tuning schema matching software using synthetic scenarios. *VLDB J.* **16**(1), 97–122 (2007) (pp. 181, 264)
- Léger, A., Nixon, L., Shvaiko, P.: On identifying knowledge processing requirements. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 928–943 (2005) (p. 360)
- Lenat, D., Guha, R.: Building Large Knowledge-Based Systems. Addison Wesley, Reading (1990) (p. 32)
- Lenzerini, M.: Data integration: a theoretical perspective. In: Proc. 21st Symposium on Principles of Database Systems (PODS), Madison, WI, USA, pp. 233–246 (2002) (pp. 10, 11, 41, 240)
- Leone, N., Greco, G., Ianni, G., Lio, V., Terracina, G., Eiter, T., Faber, W., Fink, M., Gottlob, G., Rosati, R., Lembo, D., Lenzerini, M., Ruzzi, M., Kalka, E., Nowicki, B., Staniszewski, W.: The INFORMIX system for advanced integration of incomplete and inconsistent data. In: Proc. 24th International Conference on Management of Data (SIGMOD), Baltimore, MD, USA, pp. 915–917 (2005) (p. 201)
- Lerner, B.S.: A model for compound type changes encountered in schema evolution. *ACM Trans. Database Syst.* **25**(1), 83–127 (2000) (p. 208)
- Lesk, M.: Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: Proc. 5th Annual International Conference on Systems Documentation (SIGDOC), Toronto, Canada, pp. 24–26 (1986) (pp. 99, 103)
- Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. *Dokl. Akad. Nauk SSSR* **163**(4), 845–848 (1965). In Russian. English translation in Sov. Phys. Dokl. **10**(8), 707–710 (1966) (p. 91)

- Li, W.-S., Clifton, C.: Semantic integration in heterogeneous databases using neural networks. In: Proc. 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, pp. 1–12 (1994) (pp. 118, 175, 176, 242)
- Li, W.-S., Clifton, C.: SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.* **33**(1), 49–84 (2000) (p. 242)
- Li, Y., Li, J., Zhang, D., Tang, J.: Result of ontology alignment with RiMOM at OAEI-06. In: Proc. 1st International Workshop on Ontology Matching (OM) at the 5th International Semantic Web Conference (ISWC), Athens, GA, USA, pp. 181–190 (2006) (p. 249)
- Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM: a dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.* **21**(8), 1218–1232 (2009) (pp. 181, 250)
- Lim, E.-P., Srivastava, J., Prabhakar, S., Richardson, J.: Entity identification in database integration. In: Proc. 9th International Conference on Data Engineering (ICDE), Vienna, Austria, pp. 294–301 (1993) (p. 117)
- Lin, D.: An information-theoretic definition of similarity. In: Proc. 15th International Conference of Machine Learning (ICML), Madison, WI, USA, pp. 296–304 (1998) (p. 103)
- Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce. Morgan & Claypool, San Rafael (2010) (p. 151)
- Locoro, A., David, J., Euzenat, J.: Context-based matching: design of a flexible framework and experiment. *J. Data Semant.* **2** (2013, in press) (p. 158)
- Lopez, V., Pasin, M., Motta, E.: AquaLog: an ontology-portable question answering system for the semantic web. In: Proc. 2nd European Semantic Web Conference (ESWC), Herounisou, Greece. Lecture Notes in Computer Science, vol. 3532, pp. 546–562 (2005) (p. 22)
- Lopez, V., Motta, E., Uren, V.: PowerAqua: fishing the semantic web. In: Proc. 3rd European Semantic Web Conference (ESWC), Budva, Montenegro. Lecture Notes in Computer Science, vol. 4011, pp. 393–410 (2006) (p. 22)
- Lovász, L., Plummer, M.: Matching Theory. North-Holland, Amsterdam (1986) (p. 190)
- Lovins, J.B.: Development of a stemming algorithm. *Mech. Transl. Comput. Linguist.* **11**(1), 22–31 (1968) (p. 98)
- Mädc̄he, A., Staab, S.: Measuring similarity between ontologies. In: Proc. 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Siguenza, Spain. Lecture Notes in Computer Science, vol. 2473, pp. 251–263 (2002) (pp. 123, 128)
- Mädc̄he, A., Zacharias, V.: Clustering ontology-based metadata in the semantic web. In: Proc. 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Helsinki, Finland, pp. 348–360 (2002) (p. 125)
- Mädc̄he, A., Motik, B., Silva, N., Volz, R.: MAFRA—a mapping framework for distributed ontologies. In: Proc. 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Siguenza, Spain. Lecture Notes in Computer Science, vol. 2473, pp. 235–250 (2002) (pp. 323, 324, 344)
- Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with Cupid. In: Proc. 27th International Conference on Very Large Data Bases (VLDB), Rome, Italy, pp. 48–58 (2001) (pp. 128, 210)
- Madhavan, J., Bernstein, P., Domingos, P., Halevy, A.: Representing and reasoning about mappings between domain models. In: Proc. 18th National Conference on Artificial Intelligence (AAAI), Edmonton, Canada, pp. 122–133 (2002) (pp. 31, 341)
- Madhavan, J., Bernstein, P., Doan, A.-H., Halevy, A.: Corpus-based schema matching. In: Proc. 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, pp. 57–68 (2005) (p. 250)
- Maio, P., Silva, N.: GOALS: a test-bed for ontology matching. In: Proc. 1st International Conference on Knowledge Engineering and Ontology Development (KEOD), Madeira, Portugal, pp. 293–299 (2009) (p. 266)
- Makris, K., Gioldasis, N., Bikakis, N., Christodoulakis, S.: Ontology mapping and SPARQL rewriting for querying federated RDF data sources. In: Proc. 9th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Hersonissos, Greece. Lecture Notes in Computer Science, vol. 6427, pp. 1108–1117 (2010) (p. 389)

- Malucelli, F., Ottmann, T., Pretolani, D.: Efficient labelling algorithms for the maximum noncrossing matching problem. *Discrete Appl. Math.* **47**(2), 175–179 (1993) (p. 222)
- Mao, M., Peng, Y., Spring, M.: An adaptive ontology mapping approach with neural network based constraint satisfaction. *J. Web Semant.* **8**(1), 14–25 (2010) (pp. 165, 175, 258)
- Marie, A., Gal, A.: Boosting schema matchers. In: Proc. 16th International Conference on Cooperative Information Systems (CoopIS), Monterrey, Mexico. Lecture Notes in Computer Science, vol. 5331, pp. 283–300 (2008) (pp. 267, 297)
- Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D.: ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *Proc. VLDB Endow.* **4**(12), 1438–1441 (2011) (p. 384)
- Mascardi, V., Locoro, A., Rosso, P.: Automatic ontology matching via upper ontologies: a systematic evaluation. *IEEE Trans. Knowl. Data Eng.* **22**(5), 609–623 (2010) (p. 224)
- Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: Ontology library. Deliverable D18, Wonderweb (2003) (p. 331)
- Maynard, D.: Term recognition using combined knowledge sources. PhD thesis, Manchester Metropolitan University, Manchester, UK (1999) (p. 98)
- Maynard, D., Ananiadou, S.: Term extraction using a similarity-based approach. In: Bourigault, D., Jacquemin, C., Lhomme, M.-C. (eds.) Recent Advances in Computational Terminology, pp. 261–278. Benjamins, Amsterdam (2001) (pp. 97, 98)
- McCandless, M., Hatcher, E., Gospodnetić, O.: Lucene in Action, 2nd edn. Manning Publications, Shelter Island (2010) (p. 97)
- McCann, R., Shen, W., Doan, A.-H.: Matching schemas in online communities: a web 2.0 approach. In: Proc. 24th International Conference on Data Engineering (ICDE), Cancún, Mexico, pp. 110–119 (2008) (p. 359)
- McDermott, D., Dou, D.: Representing disjunction and quantifiers in RDF. In: Proc. 1st International Semantic Web Conference (ISWC), Chia Laguna, Italy. Lecture Notes in Computer Science, vol. 2342, pp. 250–263 (2002) (p. 380)
- McGuinness, D., Pinheiro da Silva, P.: Infrastructure for web explanations. In: Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, USA. Lecture Notes in Computer Science, vol. 2870, pp. 113–129 (2003) (p. 361)
- McGuinness, D., Pinheiro da Silva, P.: Explaining answers from the semantic web: the Inference Web approach. *J. Web Semant.* **1**(4), 397–413 (2004) (p. 361)
- McGuinness, D., Fikes, R., Rice, J., Wilder, S.: An environment for merging and testing large ontologies. In: Proc. 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR), Breckenridge, CO, USA, pp. 483–493 (2000) (p. 372)
- McNeill, F.: Dynamic ontology refinement. PhD thesis, University of Edinburgh, Edinburgh, UK (2006) (p. 23)
- Mecca, G., Papotti, P., Raunich, S., Buoncristiano, M.: Concise and expressive mappings with +Spicy. *Proc. VLDB Endow.* **2**(2), 1582–1585 (2009) (p. 384)
- Medjahed, B., Bouguettaya, A.: A multilevel composability model for semantic web services. *IEEE Trans. Knowl. Data Eng.* **17**(7), 954–968 (2005) (p. 16)
- Meilicke, C.: Alignment incoherence in ontology matching. PhD thesis, Universität Mannheim, Mannheim, Germany (2011) (pp. 145, 194)
- Meilicke, C., Stuckenschmidt, H.: An efficient method for computing alignment diagnoses. In: Proc. 3rd International Conference on Web Reasoning and Rule Systems (RR), Chantilly, VA, USA, pp. 182–196 (2009) (pp. 145, 194)
- Meilicke, C., Völker, J., Stuckenschmidt, H.: Learning disjointness for debugging mappings between lightweight ontologies. In: Proc. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Acitrezza, Italy. Lecture Notes in Computer Science, vol. 5268, pp. 93–108 (2008) (p. 195)
- Meilicke, C., Stuckenschmidt, H., Tamlin, A.: Reasoning support for mapping revision. *J. Log. Comput.* **19**(5), 807–829 (2009) (pp. 50, 145)
- Meilicke, C., Garcia Castro, R., Freitas, F., van Hage, W.R., Montiel Ponsoda, E., Ribeiro de Azevedo, R., Stuckenschmidt, H., Šváb-Zamazal, O., Svátek, V., Tamlin, A., Trojahn, C.,

- Wang, S.: MultiFarm: a benchmark for multilingual ontology matching. *J. Web Semant.* **15**, 62–68 (2012) (p. 297)
- Melnik, S.: Generic Model Management: Concepts and Algorithms. Springer, Heidelberg, Germany (2004) (p. 341)
- Melnik, S., Garcia Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm. In: Proc. 18th International Conference on Data Engineering (ICDE), San Jose, CA, USA, pp. 117–128 (2002) (pp. 132, 212, 304, 308)
- Melnik, S., Rahm, E., Bernstein, P.: Developing metadata-intensive applications with Rondo. *J. Web Semant.* **1**(1), 47–74 (2003a) (p. 342)
- Melnik, S., Rahm, E., Bernstein, P.: Rondo: a programming platform for model management. In: Proc. 22nd International Conference on Management of Data (SIGMOD), San Diego, CA, USA, pp. 193–204 (2003b) (p. 342)
- Melnik, S., Bernstein, P., Halevy, A., Rahm, E.: Supporting executable mappings in model management. In: Proc. 24th International Conference on Management of Data (SIGMOD), Baltimore, MD, USA, pp. 167–178 (2005) (pp. 132, 341, 342)
- Melton, J. (ed.): Information technology—database languages—SQL. ISO standard ISO/CEI 9075:2003, ISO (2003) (p. 33)
- Mena, E., Kashyap, V., Sheth, A., Illarramendi, A.: Observer: an approach for query processing in global information systems based on interoperability between pre-existing ontologies. In: Proc. 4th International Conference on Cooperative Information Systems (CoopIS), Brussels, Belgium, pp. 14–25 (1996) (pp. 22, 201)
- Miles, A., Bechhofer, S. (eds.): SKOS: Simple Knowledge Organization System: reference. Recommendation, W3C (2009) (p. 334)
- Miller, G.: WordNet: a lexical database for English. *Commun. ACM* **38**(11), 39–41 (1995) (p. 99)
- Miller, R., Haas, L., Hernández, M.: Schema mapping as query discovery. In: Proc. 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt, pp. 77–88 (2000) (p. 383)
- Miller, R., Hernández, M., Haas, L., Yan, L., Ho, H., Fagin, R., Popa, L.: The Clio project: managing heterogeneity. *SIGMOD Rec.* **30**(1), 78–83 (2001) (p. 383)
- Milo, T., Zohar, S.: Using schema matching to simplify heterogeneous data translation. In: Proc. 24th International Conference on Very Large Data Bases (VLDB), New York, NY, USA, pp. 122–133 (1998) (p. 204)
- Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996) (p. 184)
- Mitra, P., Wiederhold, G.: Resolving terminological heterogeneity in ontologies. In: Proc. International Workshop on Ontologies and Semantic Interoperability at the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France, pp. 45–50 (2002) (p. 206)
- Mitra, P., Wiederhold, G., Jannink, J.: Semi-automatic integration of knowledge sources. In: Proc. 2nd International Conference on Information Fusion, Sunnyvale, CA, USA, pp. 572–581 (1999) (p. 205)
- Mitra, P., Wiederhold, G., Kersten, M.: A graph-oriented model for articulation of ontology interdependencies. In: Proc. 8th Conference on Extending Database Technology (EDBT), Praha, Czech Republic. Lecture Notes in Computer Science, vol. 1777, pp. 86–100 (2000) (p. 205)
- Mitra, P., Noy, N., Jaiswal, A.: Ontology mapping discovery with uncertainty. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 537–547 (2005) (pp. 141, 142, 220)
- Močan, A., Cimpian, E.: An ontology-based data mediation framework for semantic environments. *Int. J. Semantic Web Inf. Syst.* **3**(2), 69–98 (2007) (p. 370)
- Mochol, M.: The methodology for finding suitable ontology matching approaches. PhD thesis, Freie Universität Berlin, Berlin, Germany (2009) (pp. 55, 62)
- Mochol, M., Jentzsch, A.: Towards a rule-based matcher selection. In: Proc. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Acirezza, Italy. Lecture Notes in Computer Science, vol. 5268, pp. 109–119 (2008) (p. 181)
- Modica, G., Gal, A., Jamil, H.: The use of machine-generated ontologies in dynamic information seeking. In: Proc. 9th International Conference on Cooperative Information Systems (CoopIS), Trento, Italy. Lecture Notes in Computer Science, vol. 2172, pp. 433–448 (2001) (p. 209)

- Monge, A., Elkan, C.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: Proc. International Workshop on Data Mining and Knowledge Discovery at the 16th International Conference on Management of Data (SIGMOD), Tucson, AZ, USA (1997) (p. 92)
- Montanelli, S., Castano, S.: Semantically routing queries in peer-based systems: the H-Link approach. *Knowl. Eng. Rev.* **23**(1), 51–72 (2008) (p. 15)
- Montanelli, S., Bianchini, D., Aiello, C., Baldoni, R., Bolchini, C., Bonomi, S., Castano, S., Catarci, T., De Antonellis, V., Ferrara, A., Melchiori, M., Quintarelli, E., Scannapieco, M., Schreiber, F., Tanca, L.: The ESTEEM platform: enabling P2P semantic collaboration through emerging collective knowledge. *J. Intell. Inf. Syst.* **36**(2), 167–195 (2011) (p. 15)
- Mork, P., Seligman, L., Rosenthal, A., Korb, J., Wolf, C.: The harmony integration workbench. *J. Data Semant.* **XI**, 65–93 (2008) (pp. 340, 349)
- Motik, B., Patel-Schneider, P., Cuenca Grau, B. (eds.): OWL 2 web ontology language direct semantics. Recommendation, W3C (2009a) (p. 34)
- Motik, B., Patel-Schneider, P., Parsia, B. (eds.): OWL 2 web ontology language structural specification and functional-style syntax. Recommendation, W3C (2009b) (p. 34)
- Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. *J. Artif. Intell. Res.* **36**, 165–228 (2009c) (p. 390)
- Munkres, J.: Algorithms for the assignment and transportation problems. *SIAM J. Appl. Math.* **5**(1), 32–38 (1957) (p. 191)
- Musen, M., Noy, N., Shah, N., Whetzel, P., Chute, C., Story, M.-A., Smith, B.: The national center for biomedical ontology. *J. Am. Med. Inform. Assoc.* **19**(2), 190–195 (2012) (p. 4)
- Nagy, M., Vargas-Vera, M.: Towards an automatic semantic data integration: multi-agent framework approach. In: Wu, G. (ed.) Semantic Web, pp. 107–134. In-Teh, Vukovar (2010). Chap. 7 (pp. 169, 231)
- Nandi, A., Bernstein, P.: HAMSTER: using search clicklogs for schema and taxonomy matching. *Proc. VLDB Endow.* **2**(1), 181–192 (2009) (pp. 173, 255)
- Naphade, M., Smith, J., Tesic, J., Chang, S.-F., Hsu, W., Kennedy, L., Hauptmann, A., Curtis, J.: Large-scale concept ontology for multimedia. *IEEE Multimed.* **13**(3), 86–91 (2006) (p. 241)
- Navathe, S., Buneman, P.: Integrating user views in database design. *Computer* **19**(1), 50–62 (1986) (p. 112)
- Needleman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48**(3), 443–453 (1970) (p. 91)
- Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: EDUTELLA: a P2P networking infrastructure based on RDF. In: Proc. 11th International World Wide Web Conference (WWW), Honolulu, HI, USA, pp. 604–615 (2002) (p. 15)
- Nezhad, H.M., Xu, G.Y., Benatallah, B.: Protocol-aware matching of web service interfaces for adapter development. In: Proc. 19th International Conference on World Wide Web (WWW), Raleigh, NC, USA, pp. 731–740 (2010) (p. 17)
- Ngo, D.H., Bellahsene, Z.: YAM++: a multi-strategy based approach for ontology matching task. In: Proc. 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Galway, Ireland. Lecture Notes in Computer Science, vol. 7603, pp. 421–425 (2012) (p. 259)
- Ngo, D.H., Bellahsene, Z., Coletta, R.: A generic approach for combining linguistic and context profile metrics in ontology matching. In: Proc. 10th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Hersonissos, Greece, pp. 800–807 (2011) (p. 259)
- Ngomo, A.-C.N.: A time-efficient hybrid approach to link discovery. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 1–12 (2011) (p. 117)
- Ngomo, A.-C.N., Auer, S.: LIMES: a time-efficient approach for large-scale link discovery on the web of data. In: Proc. 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, pp. 2312–2317 (2011) (p. 117)

- Niepert, M., Meilicke, C., Stuckenschmidt, H.: A probabilistic-logical framework for ontology matching. In: Proc. 24th Conference on Artificial Intelligence (AAAI), Atlanta, GA, USA, pp. 1413–1418 (2010) (pp. 143, 260)
- Nikolov, A., Uren, V., Motta, E., de Roeck, A.: Integration of semantically annotated data by the KnoFuss architecture. In: Proc. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Acitrezza, Italy. Lecture Notes in Computer Science, vol. 5268, pp. 265–274 (2008) (p. 385)
- Nikolov, A., Uren, V., Motta, E., de Roeck, A.: Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In: Proc. 4th Asian Semantic Web Conference (ASWC), Bangkok, Thailand. Lecture Notes in Computer Science, vol. 5367, pp. 332–346 (2009) (pp. 115, 386)
- Niles, I., Pease, A.: Towards a standard upper ontology. In: Proc. 2nd International Conference on Formal Ontology in Information Systems (FOIS), Ogunquit, ME, USA, pp. 2–9 (2001) (p. 32)
- Niu, X., Wang, H., Wu, G., Qi, G., Yu, Y.: Evaluating the stability and credibility of ontology matching methods. In: Proc. 8th Extended Semantic Web Conference (ESWC), Hersounisou, Greece. Lecture Notes in Computer Science, vol. 6643, pp. 275–289 (2011) (p. 305)
- Nodine, M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., Unruh, A.: Active information gathering in infosleuth. *Int. J. Coop. Inf. Syst.* **9**(1–2), 3–28 (2000) (p. 201)
- Nottelmann, H., Straccia, U.: sPLMap: a probabilistic approach to schema matching. In: Proc. 27th European Conference on Information Retrieval Research (ECIR), Santiago de Compostela, Spain, pp. 81–95 (2005) (p. 240)
- Nottelmann, H., Straccia, U.: A probabilistic, logic-based framework for automated web directory alignment. In: Ma, Z. (ed.) *Soft Computing in Ontologies and the Semantic Web. Studies in Fuzziness and Soft Computing*, vol. 204, pp. 47–77. Springer, Berlin (2006) (pp. 174, 240)
- Noy, N.: Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.* **33**(4), 65–70 (2004a) (p. 201)
- Noy, N.: Tools for mapping and merging ontologies. In: Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, pp. 365–384. Springer, Berlin (2004b). Chap. 18 (p. 345)
- Noy, N., Klein, M.: Ontology evolution: not the same as schema evolution. *Knowl. Inf. Syst.* **6**(4), 428–440 (2004) (pp. 5, 39)
- Noy, N., Musen, M.: SMART: automated support for ontology merging and alignment. In: Proc. 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW), Banff, Canada (1999) (p. 209)
- Noy, N., Musen, M.: PROMPT: algorithm and tool for automated ontology merging and alignment. In: Proc. 17th National Conference on Artificial Intelligence (AAAI), Austin, TX, USA, pp. 450–455 (2000) (p. 373)
- Noy, N., Musen, M.: Anchor-PROMPT: using non-local context for semantic matching. In: Proc. International Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, USA, pp. 63–70 (2001) (p. 208)
- Noy, N., Musen, M.: Evaluating ontology-mapping tools: requirements and experience. In: Proc. 1st International Workshop on Evaluation of Ontology Tools (EON) at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Siguenza, Spain, pp. 1–14 (2002a) (pp. 289, 292)
- Noy, N., Musen, M.: PromptDiff: a fixed-point algorithm for comparing ontology versions. In: Proc. 18th National Conference on Artificial Intelligence (AAAI), Edmonton, Canada, pp. 744–750 (2002b) (pp. 5, 346)
- Noy, N., Musen, M.: The PROMPT suite: interactive tools for ontology merging and mapping. *Int. J. Hum.-Comput. Stud.* **59**(6), 983–1024 (2003) (p. 345)
- Noy, N., Musen, M.: Ontology versioning in an ontology management framework. *IEEE Intell. Syst.* **19**(4), 6–13 (2004) (p. 5)
- Noy, N., Chugh, A., Alani, H.: The CKC challenge: exploring tools for Collaborative Knowledge Construction. *IEEE Intell. Syst.* **23**(1), 64–68 (2008) (pp. 340, 392)
- Oberle, D., Volz, R., Staab, S., Motik, B.: An extensible ontology software environment. In: Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, pp. 299–319. Springer, Berlin (2004). Chap. 15

- (p. 348)
- Open Information Model, Version 1.0 (1999). <http://mdcinfo/oim/oim10.html> (p. 212)
- Oundhakar, S., Verma, K., Sivashanugam, K., Sheth, A., Miller, J.: Discovery of web services in a multi-ontology and federated registry environment. *Int. J. Web Serv. Res.* **2**(3), 1–32 (2005) (pp. 16, 218)
- Özsu, M.T., Valduriez, P.: *Principles of Distributed Database Systems*, 3rd edn. Springer, Heidelberg (2011) (p. 8)
- Pachéco, T., Pereira, A., Pires, C.E., Salgado, A.C.: Exploring web semantic knowledge and user feedback to improve ontology matching. In: Proc. 10th International Workshop on Web Semantics (WebS) at the 22nd International Conference on Database and Expert Systems Applications (DEXA), Toulouse, France, pp. 234–238 (2011) (p. 356)
- Padilha, N., Baião, F., Revoredo, K.: Ontology alignment for semantic data integration through foundational ontologies. In: Proc. 2nd International Workshop on Modeling and Reasoning for Business Intelligence (MORE-BI) at the 31st International Conference on Conceptual Modeling (ER), Florence, Italy. Lecture Notes in Computer Science, vol. 7518, pp. 172–181 (2012) (p. 130)
- Palopoli, L., Saccá, D., Terracina, G., Ursino, D.: Uniform techniques for deriving similarities of objects and subschemes in heterogeneous databases. *IEEE Trans. Knowl. Data Eng.* **15**(2), 271–294 (2003a) (p. 205)
- Palopoli, L., Terracina, G., Ursino, D.: DIKE: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Softw. Pract. Exp.* **33**(9), 847–884 (2003b) (p. 205)
- Pan, R., Ding, Z., Yu, Y., Peng, Y.: A Bayesian network approach to ontology mapping. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 563–577 (2005) (pp. 141, 219)
- Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proc. 1st International Semantic Web Conference (ISWC), Chia Laguna, Italy. Lecture Notes in Computer Science, vol. 2342, pp. 333–347 (2002) (p. 16)
- Parent, C., Spaccapietra, S.: Issues and approaches of database integration. *Commun. ACM* **41**(5), 166–178 (1998) (pp. 5, 8, 73)
- Parent, C., Spaccapietra, S.: Database integration: the key to data interoperability. In: Papazoglou, M., Spaccapietra, S., Tari, Z. (eds.) *Object-Oriented Data Modeling*, pp. 221–253. MIT Press, Cambridge (2000). Chap. 9 (pp. 147, 201)
- Parmentier, G., Bastian, F., Robinson-Rechavi, M.: Homolonto: generating homology relationships by pairwise alignment of ontologies and application to vertebrate anatomy. *Bioinformatics* **26**(14), 1766–1771 (2010) (p. 230)
- Paulheim, H., Hertling, S., Ritze, D.: Towards evaluating interactive ontology matching tools. In: Proc. 10th European Semantic Web Conference (ESWC), Montpellier, France, pp. 31–45 (2013) (p. 313)
- Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco (1988) (p. 143)
- Pedersen, T., Patwardhan, S., Michelizzi, J.: WordNet::Similarity—measuring the relatedness of concepts. In: Proc. 19th National Conference on Artificial Intelligence (AAAI), San Jose, CA, USA, pp. 1024–1025 (2004) (p. 106)
- Peukert, E., Eberius, J., Rahm, E.: AMC: a framework for modelling and comparing matching systems as matching processes. In: Proc. 27th International Conference on Data Engineering (ICDE), Hannover, Germany, pp. 1304–1307 (2011) (pp. 268, 356)
- Peukert, E., Eberius, J., Rahm, E.: A self-configuring schema matching system. In: Proc. 28th International Conference on Data Engineering (ICDE), Washington, DC, USA, pp. 306–317 (2012) (pp. 181, 268)
- Pinheiro da Silva, P., McGuinness, D., Fikes, R.: A proof markup language for semantic web services. *Inf. Sci.* **31**(4), 381–395 (2006) (p. 361)
- Pinto da Silva, N.A.: Multi-dimensional service-oriented ontology mapping. PhD thesis, Universidade de Trás-os-Montes e Alto Douro, Villa Real, Portugal (2004) (pp. 323, 344)

- Po, L., Bergamaschi, S.: Automatic lexical annotation applied to the SCARLET ontology matcher. In: Proc. 2nd International Conference on Intelligent Information and Database Systems (ACI-IDS), Hue City, Vietnam, pp. 144–153 (2010) (p. 224)
- Porter, M.: An algorithm for suffix stripping. *Program* **14**(3), 130–137 (1980) (p. 98)
- Porzel, R., Malaka, R.: A task-based approach for ontology evaluation. In: Proc. Workshop on Ontology Learning and Population at the 16th European Conference on Artificial Intelligence (ECAI), Valencia, Spain (2004) (p. 316)
- Preece, A., Hui, K.-Y., Gray, A., Marti, P., Bench-Capon, T., Jones, D., Cui, Z.: The KRAFT architecture for knowledge fusion and transformation. *Knowl.-Based Syst.* **13**(2–3), 113–120 (2000) (p. 201)
- Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL query language for RDF. Recommendation, W3C (2008) (p. 42)
- Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. *Arch. Math. Phys.* **27**, 742–744 (1918) (p. 219)
- Qazvinian, V., Abolhassani, H., Haeri (Hossein), S., Hariri, B.B.: Evolutionary coincidence-based ontology mapping extraction. *Expert Syst.* **25**(3), 221–236 (2008) (pp. 185, 222)
- Qi, G., Ji, Q., Haase, P.: A conflict-based operator for mapping revision. In: Proc. 8th International Semantic Web Conference (ISWC), Chantilly, VA, USA. Lecture Notes in Computer Science, vol. 5823, pp. 521–536 (2009) (p. 148)
- Qu, Y., Hu, W., Chen, G.: Constructing virtual documents for ontology matching. In: Proc. 15th International World Wide Web Conference (WWW), Edinburgh, UK, pp. 23–31 (2006) (pp. 93, 248)
- Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, Menlo Park (1993) (p. 178)
- Quix, C., Kensche, D., Li, X.: Generic schema merging. In: Proc. 19th International Conference on Advanced Information Systems Engineering (CAiSE), Trondheim, Norway, pp. 127–141 (2007a) (p. 222)
- Quix, C., Kensche, D., Li, X.: Matching of ontologies with XML schemas using a generic meta-model. In: Proc. 6th of International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Vilamoura, Portugal, pp. 1081–1098 (2007b) (p. 222)
- Quix, C., Gal, A., Sagi, T., Kensche, D.: An integrated matching system GeRoMeSuite and SMB: results for OAEI 2010. In: Proc. 5th International Workshop on Ontology Matching (OM) at the 9th International Semantic Web Conference (ISWC), Shanghai, China, pp. 166–171 (2010) (p. 223)
- Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. *VLDB J.* **10**(4), 334–350 (2001) (pp. 46, 73, 76, 77, 78, 83, 107, 201)
- Raunich, S., Rahm, E.: ATOM: Automatic Target-driven Ontology Merging. In: Proc. 27th International Conference on Data Engineering (ICDE), Hannover, Germany, pp. 1276–1279 (2011) (p. 343)
- Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI), Montréal, Canada, pp. 448–453 (1995) (p. 102)
- Resnik, P.: Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.* **11**, 95–130 (1999) (pp. 102, 125)
- Richardson, M., Domingos, P.: Markov logic networks. *Mach. Learn.* **62**(1–2), 107–136 (2006) (p. 143)
- Riedel, S.: Improving the accuracy and efficiency of MAP inference for Markov logic. In: Proc. 24th Conference in Uncertainty in Artificial Intelligence (UAI), Helsinki, Finland, pp. 468–475 (2008) (p. 260)
- Ritze, D., Paulheim, H.: Towards an automatic parameterization of ontology matching tools based on example mappings. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 37–48 (2011) (p. 182)

- Ritze, D., Meilicke, C., Šváb-Zamazal, O., Stuckenschmidt, H.: A pattern-based ontology matching approach for detecting complex correspondences. In: Proc. 4th International Workshop on Ontology Matching (OM) at the 8th International Semantic Web Conference (ISWC), Chantilly, VA, USA, pp. 25–36 (2009) (p. 130)
- Rivero, C., Hernández, I., Ruiz, D., Corchuelo, R.: Generating SPARQL executable mappings to integrate ontologies. In: Proc. 30th International Conference on Conceptual Modeling (ER), Brussels, Belgium. Lecture Notes in Computer Science, vol. 6998, pp. 118–131 (2011) (p. 382)
- Robertson, D.: A lightweight coordination calculus for agent systems. In: Revised Selected Papers from 2nd International Workshop on Declarative Agent Languages and Technologies (DALT), New York, NY, USA. Lecture Notes in Computer Science, vol. 3476, pp. 183–197 (2004) (p. 264)
- Robertson, S., Spärck Jones, K.: Relevance weighting of search terms. *J. Am. Soc. Inf. Sci.* **27**(3), 129–146 (1976) (p. 94)
- Robertson, D., Giunchiglia, F., van Harmelen, F., Marchese, M., Sabou, M., Schorlemmer, M., Shadbolt, N., Siebes, R., Sierra, C., Walton, C., Dasmahapatra, S., Dupplaw, D., Lewis, P., Yatskevich, M., Kotoulas, S., de Pinnick, A.P., Loizou, A.: Open knowledge semantic webs through peer-to-peer interaction. Technical Report DIT-06-034, University of Trento (2006) (p. 17)
- Roddick, J.: A survey of schema versioning issues for database systems. *Inf. Softw. Technol.* **37**(7), 383–393 (1995) (p. 5)
- Rousset, M.-C., Adjiman, P., Chatalic, P., Goasdoué, F., Simon, L.: Somewhere in the semantic web. In: Proc. 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SofSem), Merin, Czech Republic. Lecture Notes in Computer Science, vol. 3831, pp. 84–99 (2006) (pp. 14, 15)
- Roussej, C., Corcho, Ó., Vilches Blázquez, L.M.: A catalogue of OWL ontology antipatterns. In: Proc. 5th International Conference on Knowledge Capture (K-CAP), Redondo Beach, CA, USA, pp. 205–206 (2009) (p. 130)
- Russell, S., Norvig, P.: Artificial Intelligence: a Modern Approach. Prentice Hall, Englewood Cliffs (1995) (pp. 141, 236)
- Sabou, M., Lopez, V., Motta, E.: Ontology selection for the real semantic web: how to cover the Queen birthday dinner? In: Proc. 15th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Praha, Czech Republic. Lecture Notes in Computer Science, vol. 4248, pp. 96–111 (2006) (pp. 21, 22)
- Sabou, M., d'Aquin, M., Motta, E.: Exploring the semantic web as background knowledge for ontology matching. *J. Data Semant.* **XI**, 156–190 (2008a) (p. 223)
- Sabou, M., d'Aquin, M., Motta, E.: SCARLET: Semantic RelAtion discoveRy by harvesting on-LInE onTologies. In: Proc. 5th European Semantic Web Conference (ESWC), Tenerife, Spain. Lecture Notes in Computer Science, vol. 5021, pp. 854–858 (2008b) (p. 223)
- Saint-Onge, D.: Detecting and correcting malapropisms with lexical chains. Master's thesis, University of Toronto, Toronto, Canada (1995) (p. 106)
- Saleem, K., Bellahsene, Z., Hunt, E.: PORSCHE: Performance ORiented SCHEma mediation. *Inf. Sci.* **33**(7–8), 637–657 (2008) (p. 228)
- Salton, G.: The SMART Retrieval System: Experiments in Automatic Information Processing. Prentice Hall, Englewood Cliffs (1971) (p. 93)
- Salton, G., McGill, M.: Introduction to Modern Information Retrieval. McGraw-Hill, New York (1983) (p. 93)
- Sarasua, C., Simperl, E., Noy, N.: CrowdMAP: crowdsourcing ontology alignment with micro-tasks. In: Proc. 11th International Semantic Web Conference (ISWC), Boston, MA, USA. Lecture Notes in Computer Science, vol. 7649, pp. 525–541 (2012) (p. 359)
- Sarma, A.D., Dong, X., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: Proc. 27th International Conference on Management of Data (SIGMOD), Vancouver, BC, Canada, pp. 861–874 (2008) (p. 357)
- Sayyadian, M., Lee, Y., Doan, A.-H., Rosenthal, A.: Tuning schema matching software using synthetic scenarios. In: Proc. 31st International Conference on Very Large Data Bases (VLDB),

- Trondheim, Norway, pp. 994–1005 (2005) (p. 264)
- Scharffe, F.: Representing correspondence patterns with expressive ontology alignments. PhD thesis, University of Innsbruck, Innsbruck, Austria (2008) (p. 129)
- Scharffe, F., Euzenat, J.: Linked data meets ontology matching: enhancing data linking through ontology alignments. In: Proc. 3rd International Conference on Knowledge Engineering and Ontology Development (KEOD), Paris, France, pp. 279–284 (2011) (pp. 12, 13, 115, 385)
- Scharffe, F., Fensel, D.: Correspondence patterns for ontology mediation. In: Proc. 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Acitrezza, Italy. Lecture Notes in Computer Science, vol. 5268, pp. 83–92 (2008) (p. 129)
- Schuh, R.: Biological Systematics: Principles and Applications. Cornell University Press, Ithaca (1999) (p. 28)
- Schulten, E., Akkermans, H., Botquin, G., Dorr, M., Guarino, N., Lopes, N., Sadeh, N.: Call for participants: the e-commerce product classification challenge. IEEE Intell. Syst. **16**(4), 86–c3 (2001) (p. 9)
- Seligman, L., Mork, P., Halevy, A., Smith, K., Carey, M., Chen, K., Wolf, C., Madhavan, J., Kannan, A., Burdick, D.: OpenII: an open source information integration toolkit. In: Proc. 29th International Conference on Management of Data (SIGMOD), Indianapolis, IN, USA, pp. 1057–1060 (2010) (pp. 5, 349)
- Serafini, L., Tamlilin, A.: DRAGO: distributed reasoning architecture for the semantic web. In: Proc. 2nd European Semantic Web Conference (ESWC), Herounisous, Greece. Lecture Notes in Computer Science, vol. 3532, pp. 361–376 (2005) (p. 390)
- Serafini, L., Stuckenschmidt, H., Wache, H.: A formal investigation of mapping language for terminological knowledge. In: Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh, UK, pp. 576–581 (2005) (p. 322)
- Shadbolt, N., O’Hara, K., Salvadores, M., Alani, H.: EGovernment. In: Domingue, J., Fensel, D., Hendler, J. (eds.) Handbook of Semantic Web Technologies, pp. 840–900. Springer, Berlin (2011) (p. 12)
- Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976) (p. 167)
- Sheth, A., Larson, J.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. **22**(3), 183–236 (1990) (pp. 5, 8, 37, 46)
- Sheth, A., Larson, J., Cornelio, A., Navathe, S.: A tool for integrating conceptual schemas and user views. In: Proc. 4th International Conference on Data Engineering (ICDE), Los Angeles, CA, USA, pp. 176–183 (1988) (p. 113)
- Shvaiko, P.: Iterative schema-based semantic matching. PhD thesis, University of Trento, Trento, Italy (2006) (pp. 145, 146)
- Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. J. Data Semant. **IV**, 146–171 (2005) (pp. 73, 76, 201)
- Shvaiko, P., Euzenat, J.: Ten challenges for ontology matching. In: Proc. 7th of International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Monterrey, Mexico. Lecture Notes in Computer Science, vol. 5332, pp. 1164–1182 (2008) (p. 401)
- Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. IEEE Trans. Knowl. Data Eng. **25**(1), 158–176 (2013) (pp. ix, 401)
- Shvaiko, P., Giunchiglia, F., Pinheiro da Silva, P., McGuinness, D.: Web explanations for semantic heterogeneity discovery. In: Proc. 2nd European Semantic Web Conference (ESWC), Herounisous, Greece. Lecture Notes in Computer Science, vol. 3532, pp. 303–317 (2005) (pp. 360, 361)
- Shvaiko, P., Giunchiglia, F., Schorlemmer, M., McNeill, F., Bundy, A., Marchese, M., Yatskevich, M., Zaihrayeu, I., Ho, B., Lopez, V., Sabou, M., Abian, J., Siebes, R., Kotoulas, S.: Dynamic ontology matching: a survey. Deliverable 3.1, OpenKnowledge (2006) (p. 15)
- Shvaiko, P., Giunchiglia, F., Yatskevich, M.: Semantic matching with S-Match. In: De Virgilio, R., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management, pp. 183–202. Springer, Berlin (2009) (p. 214)

- Shvaiko, P., Farazi, F., Maltese, V., Ivanyukovich, A., Rizzi, V., Ferrari, D., Ucelli, G.: Trentino government linked open geo-data: a case study. In: Proc. 11th International Semantic Web Conference (ISWC), Boston, MA, USA. Lecture Notes in Computer Science, vol. 7650, pp. 196–211 (2012) (p. 12)
- Silva, N., Maio, P., Rocha, J.: An approach to ontology mapping negotiation. In: Proc. International Workshop on Integrating Ontologies at the 3rd International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 54–60 (2005) (p. 169)
- Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. *J. Web Semant.* **5**(2), 51–53 (2007) (pp. 213, 260, 390)
- Smets, P.: The combination of evidence in the transferable belief model. *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(5), 447–458 (1990) (p. 168)
- Smith, T., Waterman, M.: Identification of common molecular subsequences. *J. Mol. Biol.* **147**(1), 195–197 (1981) (p. 92)
- Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L., Eilbeck, K., Ireland, A., Mungall, C., Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.-A., Scheuermann, R., Shah, N., Whetzel, P., Lewis, S.: The OBO foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat. Biotechnol.* **25**, 1251–1255 (2007) (p. 4)
- Smith, K., Morse, M., Mork, P., Li, M., Rosenthal, A., Allen, D., Seligman, L.: The role of schema matching in large enterprises. In: Proc. 4th Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA (2009) (p. 349)
- Smith, K., Mork, P., Seligman, L., Leveille, P., Yost, B., Li, M.H., Wolf, C.: Unity: speeding the creation of community vocabularies for information integration and reuse. In: Proc. 10th International Conference on Information Reuse and Integration (IRI), Las Vegas, NV, USA, pp. 129–135 (2011) (p. 349)
- Smolka, G.: Feature constraints logics for unification grammars. *J. Log. Program.* **12**(1), 324–343 (1992) (p. 328)
- Sotnykova, A., Vangenot, C., Cullot, N., Bennacer, N., Aufaure, M.-A.: Semantic mappings in description logics for spatio-temporal database schema integration. *J. Data Semant.* **III**, 143–167 (2005) (p. 147)
- Spaccapietra, S., Parent, C.: Conflicts and correspondence assertions in interoperable databases. *SIGMOD Rec.* **20**(4), 49–54 (1991) (p. 5)
- Spiliopoulos, V., Vouros, G., Karkaletsis, V.: On the discovery of subsumption relations for the alignment of ontologies. *J. Web Semant.* **8**(1), 69–88 (2010) (pp. 75, 173, 174, 177, 178, 257)
- Spohr, D., Hollink, L., Cimiano, P.: A machine learning approach to multilingual and cross-lingual ontology matching. In: Proc. 10th International Semantic Web Conference (ISWC), Bonn, Germany. Lecture Notes in Computer Science, vol. 7031, pp. 665–680 (2011) (p. 105)
- Staab, S., Stuckenschmidt, H. (eds.): Semantic Web and Peer-to-Peer. Springer, Heidelberg (2006) (p. 14)
- Staab, S., Studer, R.: Handbook on Ontologies. International Handbooks on Information Systems. Springer, Berlin (2004) (p. 33)
- Stoilos, G., Stamou, G., Kollias, S.: A string metric for ontology alignment. In: Proc. 4th International Semantic Web Conference (ISWC), Galway, Ireland. Lecture Notes in Computer Science, vol. 3729, pp. 624–637 (2005) (p. 93)
- Straccia, U., Troncy, R.: oMAP: combining classifiers for aligning automatically OWL ontologies. In: Proc. 6th International Conference on Web Information Systems Engineering (WISE), New York, NY, USA, pp. 133–147 (2005) (p. 244)
- Straccia, U., Troncy, R.: Towards distributed information retrieval in the semantic web: query reformulation using the oMAP framework. In: Proc. 3rd European Semantic Web Conference (ESWC), Budva, Montenegro. Lecture Notes in Computer Science, vol. 4011, pp. 378–392 (2006) (pp. 173, 244)
- Stroulia, E., Wang, Y.: Structural and semantic matching for assessing web-service similarity. *Int. J. Coop. Inf. Syst.* **14**(4), 407–438 (2005) (p. 217)
- Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.): Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization. Lecture Notes in Computer Science, vol. 5445.

- Springer, Berlin (2009) (p. 151)
- Stumme, G., Mädche, A.: FCA-Merge: bottom-up merging of ontologies. In: Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, USA, pp. 225–234 (2001) (p. 234)
- Su, X., Gulla, J.A.: An information retrieval approach to ontology mapping. *Data Knowl. Eng.* **58**(1), 47–69 (2006) (p. 251)
- Su, W., Wang, J., Lochovsky, F.: Holistic schema matching for web query interfaces. In: Proc. 10th Conference on Extending Database Technology (EDBT), Munich, Germany. Lecture Notes in Computer Science, vol. 3896, pp. 77–94 (2006) (pp. 221, 293)
- Suarez Meza, L.J., Potosi, L.A.R., Corrales, J.C., Steller, L.A.: Service discovery in ubiquitous computing environments. In: Proc. 6th International Conference on Internet and Web Applications and Services (ICIW), St. Maarten, The Netherlands, pp. 1–9 (2011) (p. 20)
- Suárez-Figueroa, M., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.): *Ontology Engineering in a Networked World*. Springer, Heidelberg (2012) (pp. 56, 68)
- Suchanek, F., Abiteboul, S., Senellart, P.: PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *Proc. VLDB Endow.* **5**(3), 157–168 (2012) (p. 262)
- Sun, A., Lin, E.-P.: Hierarchical text classification and evaluation. In: Proc. 1st International Conference on Data Mining (ICDM), San Jose, CA, USA, pp. 521–528 (2001) (p. 307)
- Šváb-Zamazal, O.: Pattern-based ontology matching and ontology alignment evaluation. PhD thesis, University of Economics, Prague, Czech Republic (2010) (p. 130)
- Taheriyan, M., Knoblock, C., Szekely, P., Ambite, J.L.: Rapidly integrating services into the linked data cloud. In: Proc. 11th International Semantic Web Conference (ISWC), Boston, MA, USA. Lecture Notes in Computer Science, vol. 7649, pp. 559–574 (2012) (p. 23)
- Tai, K.-C.: The tree-to-tree correction problem. *J. ACM* **26**(3), 422–433 (1979) (p. 127)
- Tang, J., Li, J., Liang, B., Huang, X., Li, Y., Wang, K.: Using Bayesian decision for ontology mapping. *J. Web Semant.* **4**(1), 243–262 (2006) (p. 249)
- Taylor, A.: *Social Choice and the Mathematics of Manipulation*. Cambridge University Press, Cambridge (2005) (p. 167)
- Thaler, S., Simperl, E.P.B., Siorpaes, K.: SpotTheLink: playful alignment of ontologies. In: Proc. 26th ACM Symposium on Applied Computing (SAC), Tai Chung, Taiwan, pp. 1711–1712 (2011) (p. 359)
- Thayavaram, U., Doshi, P.: Optima results for OAEI 2011. In: Proc. 6th International Workshop on Ontology Matching (OM) at the 10th International Semantic Web Conference (ISWC), Bonn, Germany, pp. 204–211 (2011) (pp. 138, 257)
- Thayavaram, U., Doshi, P.: Improved convergence of iterative ontology alignment using block-coordinate descent. In: Proc. 26th Conference on Artificial Intelligence (AAAI), Toronto, Canada, pp. 150–156 (2012) (p. 257)
- Ting, K.M., Witten, I.: Issues in stacked generalization. *J. Artif. Intell. Res.* **10**, 271–289 (1999) (pp. 183, 184)
- Tong, H., Faloutsos, C., Pan, J.-Y.: Fast random walk with restart and its applications. In: Proc. 6th IEEE International Conference on Data Mining (ICDM), Hong Kong, China, pp. 613–622 (2006) (p. 242)
- Tordai, A.: On combining alignment techniques. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands (2012) (pp. 65, 193)
- Tordai, A., van Ossenbruggen, J., Wielinga, B.: Let's agree to disagree: on the evaluation of vocabulary alignment. In: Proc. 6th International Conference on Knowledge Capture (K-CAP), Banff, Canada, pp. 65–72 (2011) (pp. 296, 311)
- Tournaire, R.: *Découverte automatique de correspondances entre ontologies*. PhD thesis, Université de Grenoble, Grenoble, France (2010) (p. 299)
- Tournaire, R., Petit, J.-M., Rousset, M.-C., Termier, A.: Discovery of probabilistic mappings between taxonomies: principles and experiments. *J. Data Semant.* **XV**, 66–101 (2011) (pp. 173, 177, 178, 242)
- Trillo, R., Gracia, J., Espinoza, M., Mena, E.: Discovering the semantics of user keywords. *J. Univers. Comput. Sci.* **13**(12), 1908–1935 (2007) (p. 226)

- Trojahn, C., Moraes, M., Quaresma, P., Vieira, R.: A cooperative approach for composite ontology mapping. *J. Data Semant.* **X**, 237–263 (2008) (p. 171)
- Trojahn, C., Meilicke, C., Euzenat, J., Stuckenschmidt, H.: Automating OAEI campaigns (first report). In: Proc. International Workshop on Evaluation of Semantic Technologies (WEST) at the 9th International Semantic Web Conference (ISWC), Shanghai, China (2010a) (p. 291)
- Trojahn, C., Quaresma, P., Vieira, R.: An API for multilingual ontology matching. In: Proc. 7th Language Resources and Evaluation Conference (LREC), Valletta, Malta, pp. 3830–3835 (2010b) (p. 105)
- Trojahn, C., Euzenat, J., Tamma, V., Payne, T.: Argumentation for reconciling agent ontologies. In: Elai, A., Kona, M., Orgun, M. (eds.) *Semantic Agent Systems*, pp. 89–111. Springer, New York (2011). Chap. 5 (pp. 19, 169, 360)
- Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Proc. 3rd International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Computer Science, vol. 4130, pp. 292–297. Springer, Seattle (2006) (p. 213)
- Tu, K., Yu, Y.: CMC: combining multiple schema-matching strategies based on credibility prediction. In: Proc. 10th International Conference on Database Systems for Advanced Applications (DASFAA), Beijing, China. Lecture Notes in Computer Science, vol. 3453, pp. 888–893 (2005) (p. 343)
- Tverski, A.: Features of similarity. *Psychol. Rev.* **84**(2), 327–352 (1977) (p. 86)
- Udrea, O., Getoor, L., Miller, R.: Leveraging data and structure in ontology integration. In: Proc. 26th International Conference on Management of Data (SIGMOD), Beijing, China, pp. 449–460 (2007) (p. 253)
- Uschold, M.: Achieving semantic interoperability using RDF and OWL—v4 (2005). <http://lists.w3.org/Archives/Public/public-swbp-wg/2005Sep/att-0027/SemanticII-v4.htm> (p. 324)
- Uschold, M., Gruninger, M.: Ontologies and semantics for seamless connectivity. *SIGMOD Rec.* **33**(4), 58–64 (2004) (p. 26)
- Vaccari, L., Shvaiko, P., Marchese, M.: A geo-service semantic integration in spatial data infrastructures. *Int. J. Spat. Data Infrastruct. Res.* **4**, 24–51 (2009) (p. 14)
- Vaccari, L., Shvaiko, P., Pane, J., Besana, P., Marchese, M.: An evaluation of ontology matching in geo-service applications. *GeoInformatica* **16**(1), 31–66 (2012) (pp. 215, 299)
- Valtchev, P.: Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets. Thèse d'informatique, Université Grenoble 1, Grenoble, France (1999) (pp. 95, 108, 109, 110, 119, 120, 126, 163)
- Valtchev, P., Euzenat, J.: Dissimilarity measure for collections of objects and values. In: Proc. 2nd Symposium on Intelligent Data Analysis (IDA), London, UK. Lecture Notes in Computer Science, vol. 1280, pp. 259–272 (1997) (pp. 108, 124)
- van der Meij, L., Isaac, A., Zinn, C.: A web-based repository service for vocabularies and alignments in the cultural heritage domain. In: Proc. 7th Extended Semantic Web Conference (ESWC), Heraklion, Greece. Lecture Notes in Computer Science, vol. 6088, pp. 394–409 (2010) (p. 394)
- van Eijk, R., de Boer, F., van de Hoek, W., Meyer, J.-J.: On dynamically generated ontology translators in agent communication. *Int. J. Intell. Syst.* **16**(5), 587–607 (2001) (p. 19)
- van Gendt, M., Isaac, A., van der Meij, L., Schlobach, S.: Semantic web techniques for multiple views on heterogeneous collections: a case study. In: Proc. 10th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Alicante, Spain, pp. 426–437 (2006) (p. 6)
- van Hage, W.R.: Evaluating ontology alignment techniques. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands (2007) (p. 310)
- van Ham, F., van Wijk, J.: Interactive visualization of small world graphs. In: Proc. 10th Symposium on Information Visualization (InfoVis), Austin, TX, USA, pp. 199–206 (2004) (p. 373)
- van Rijsbergen, C.J.K.: *Information Retrieval*. Butterworths, London (1975). <http://www.dcs.gla.ac.uk/Keith/Preface.html> (p. 301)
- Vapnik, V.: *The Nature of Statistical Learning Theory*, 2nd edn. Springer, New York (2000) (p. 177)

- Vargas-Vera, M., Motta, E.: AQUA: ontology-based question answering system. In: Proc. 3rd Mexican International Conference on Artificial Intelligence (MICAI), Mexico, Mexico, pp. 468–477 (2004) (p. 22)
- Vázquez-Naya, J.M., Romero, M.M., Loureiro, J.P., Munteanu, C., Sierra, A.P.: Improving ontology alignment through genetic algorithms. In: Pose, M.G., Cebrián, D.R. (eds.) *Soft Computing Methods for Practical Environment Solutions: Techniques and Studies*, pp. 1283–1289. IGI Global, Hershey (2010) (p. 185)
- Velegakis, Y., Miller, R., Popa, L.: Preserving mapping consistency under schema changes. *VLDB J.* **13**(3), 274–293 (2004a) (p. 395)
- Velegakis, Y., Miller, R., Popa, L., Mylopoulos, J.: ToMAS: a system for adapting mappings while schemas evolve. In: Proc. 20th International Conference on Data Engineering (ICDE), Boston, MA, USA, p. 862 (2004b) (p. 395)
- Visser, P., Jones, D., Bench-Capon, T., Shave, M.: Assessing heterogeneity by classifying ontology mismatches. In: Proc. 1st International Conference on Formal Ontology in Information Systems (FOIS), Trento, Italy, pp. 148–162 (1998) (p. 38)
- Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: Proc. 8th International Semantic Web Conference (ISWC), Chantilly, VA, USA. Lecture Notes in Computer Science, vol. 5823, pp. 650–665 (2009) (p. 386)
- von Hippel, E.: Democratizing Innovation. MIT Press, Cambridge (2005) (p. 353)
- Vouros, G., Kotis, K.: Extending HCONE-merge by approximating the intended interpretations of concepts iteratively. In: Proc. 2nd European Semantic Web Conference (ESWC), Herounisous, Greece. Lecture Notes in Computer Science, vol. 3532, pp. 198–210 (2005) (p. 215)
- Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information—a survey of existing approaches. In: Proc. International Workshop on Ontologies and Information Sharing at the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, USA, pp. 108–117 (2001) (pp. 5, 37, 73)
- Wang, J., Gasser, L.: Mutual online ontology alignment. In: Proc. Workshop on Ontologies in Agent Systems (OAS) at the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Bologna, Italy (2002) (p. 19)
- Wang, T., Pottinger, R.: SeMap: a generic mapping construction system. In: Proc. 11th International Conference on Extending Database Technology (EDBT), Nantes, France, pp. 97–108 (2008) (p. 253)
- Wang, P., Xu, B.: Debugging ontology mapping: a static approach. *Comput. Inform.* **27**(1), 21–36 (2008) (p. 229)
- Wang, P., Xu, B.: An effective similarity propagation method for matching ontologies without sufficient or regular linguistic information. In: Proc. 4th Asian Semantic Web Conference (ASWC), Shanghai, China. Lecture Notes in Computer Science, vol. 5926, pp. 105–119 (2009) (p. 229)
- Wang, J., Wen, J.-R., Lochovsky, F., Ma, W.-Y.: Instance-based schema matching for web databases by domain-specific query probing. In: Proc. 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, pp. 408–419 (2004) (p. 239)
- Wang, J., Ding, Z., Jiang, C.: GAOM: Genetic Algorithm based Ontology Matching. In: Proc. 1st IEEE Asia-Pacific Services Computing Conference (APSCC), GuangZhou, China, pp. 617–620 (2006) (p. 185)
- Wang, Y., Liu, W., Bell, D.: Combining uncertain outputs from multiple ontology matchers. In: Proc. 1st International Conference on Scalable Uncertainty Management (SUM), Washington, DC, USA, pp. 201–214 (2007) (p. 169)
- Wang, P., Zhou, Y., Xu, B.: Matching large ontologies based on reduction anchors. In: Proc. 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, pp. 2343–2348 (2011) (pp. 152, 229)
- Wiesman, F., Roos, N., Vogt, P.: Automatic ontology mapping for agent communication. In: Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Bologna, Italy, pp. 563–564 (2002) (p. 19)

- Wimmer, M., Seidl, M., Brosch, P., Kargl, H., Kappel, G.: On realizing a framework for self-tuning mappings. In: Proc. 47th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS), Zürich, Switzerland, pp. 1–16 (2009) (p. 256)
- Winkler, W.: The state of record linkage and current research problems. Technical Report 99/04, Statistics of Income Division, Internal Revenue Service Publication (1999) (p. 92)
- Witten, I., Frank, E., Hall, M.: Data Mining: Practical Machine Learning Tools and Techniques, 3rd edn. Morgan Kaufmann, Waltham (2011) (p. 179)
- Wolpert, D.: Stacked generalization. *Neural Netw.* **5**(2), 241–259 (1992) (p. 182)
- Wooldridge, M.: Reasoning About Rational Agents. MIT Press, Cambridge (2000) (p. 49)
- Wu, Z., Palmer, M.: Verb semantics and lexical selection. In: Proc. 32nd Annual Meeting of the Association for Computational Linguistics (ACL), Las Cruces, NM, USA, pp. 133–138 (1994) (p. 125)
- Wu, W., Yu, C., Doan, A., Meng, W.: An interactive clustering-based approach to integrating source query interfaces on the deep web. In: Proc. 23rd International Conference on Management of Data (SIGMOD), Paris, France, pp. 95–106 (2004) (p. 246)
- Wu, W., Doan, A., Yu, C.: WebIQ: learning from the web to match deep-web query interfaces. In: Proc. 22nd International Conference on Data Engineering (ICDE), Atlanta, GA, USA, p. 44 (2006) (p. 247)
- Xu, L., Embley, D.: Discovering direct and indirect matches for schema elements. In: Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA), Kyoto, Japan, pp. 39–46 (2003) (pp. 178, 179, 245)
- Yager, R.: On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Trans. Syst. Man Cybern.* **18**(1), 183–190 (1988) (pp. 165, 166)
- Yager, R.: Families of OWA operators. *Fuzzy Sets Syst.* **59**, 125–148 (1993) (p. 165)
- Zadeh, L.: Book review: a mathematical theory of evidence. *AI Mag.* **5**(3), 81–83 (1984) (p. 168)
- Zaihrayeu, I.: Towards peer-to-peer information management systems. PhD thesis, University of Trento, Trento, Italy (2006) (pp. 13, 15)
- Zaiss, K., Conrad, S., Vater, S.: A benchmark for testing instance-based ontology matching methods. In: Proc. 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Posters and Demos, Lisbon, Portugal (2010) (p. 299)
- Zanobini, S.: Semantic coordination: the model and an application to schema matching. PhD thesis, University of Trento, Trento, Italy (2006) (p. 83)
- Zhang, S., Bodenreider, O.: Experience in aligning anatomical ontologies. *Int. J. Semantic Web Inf. Syst.* **3**(2), 1–26 (2007) (p. 223)
- Zhang, S., Mork, P., Bodenreider, O.: Lessons learned from aligning two representations of anatomy. In: Proc. 13th International Conference on the Principles of Knowledge Representation and Reasoning Conference (KR), Whistler, Canada, pp. 555–560 (2004) (p. 296)
- Zhdanova, A., Shvaiko, P.: Community-driven ontology matching. In: Proc. 3rd European Semantic Web Conference (ESWC), Budva, Montenegro. Lecture Notes in Computer Science, vol. 4011, pp. 34–49 (2006) (pp. 347, 358, 391)
- Zhdanova, A., Krummenacher, R., Henke, J., Fensel, D.: Community-driven ontology management: DERI case study. In: Proc. 4th International Conference on Web Intelligence (WI), Compiegne, France, pp. 73–79 (2005) (p. 15)
- Zimmermann, A.: Logical formalisms for agreement technologies. In: Ossowski, S. (ed.) *Agreement Technologies*, pp. 69–82. Springer, Heidelberg (2013). Chap. 5 (p. 50)
- Zimmermann, A., Euzenat, J.: Three semantics for distributed systems and their relations with alignment composition. In: Proc. 5th International Semantic Web Conference (ISWC), Athens, GA, USA. Lecture Notes in Computer Science, vol. 4273, pp. 16–29 (2006) (pp. 50, 322)
- Zimmermann, A., Le Duc, C.: Reasoning on a network of aligned ontologies. In: Proc. 2nd International Conference on Web Reasoning and Rule Systems (RR), Karlsruhe, Germany, pp. 43–57 (2008) (p. 390)
- Zimmermann, A., Krötzsch, M., Euzenat, J., Hitzler, P.: Formalizing ontology alignment and its operations with category theory. In: Proc. 4th International Conference on Formal Ontology in Information Systems (FOIS), Baltimore, MD, USA, pp. 277–288 (2006) (pp. 41, 49)

- Zohar, S.: Schema-based data translation. Master's thesis, Tel-Aviv University, Tel-Aviv, Israel (1997) (p. [204](#))
- Zurawski, M., Smaill, A., Robertson, D.: Bounded ontological consistency for scalable dynamic knowledge infrastructures. In: Proc. 3rd Asian Semantic Web Conference (ASWC), Bangkok, Thailand. Lecture Notes in Computer Science, vol. 5367, pp. 212–226 (2008) (p. [195](#))

Index¹

Symbols

= (property assignment), 35
 C (ontology classes), 35
 D (domain of interpretation), 36
 $F(\cdot, \cdot)$ (fallout), 303
 $H(\cdot, \cdot)$ (Hamming distance), 301
 I (ontology individuals), 35
 $I(\cdot)$ (interpretation function), 36
 \mathcal{M} (set of models), 37
 $M(\cdot, \cdot)$ (miss), 304
 $M_\alpha(\cdot, \cdot)$ (F-measure), 304
 $O(\cdot, \cdot)$ (overall), 305
 $P(\cdot, \cdot)$ (precision), 302
 $Q_L(\cdot)$ (ontology entities), 43
 R

- ontology relations, 35
- reference alignment, 300

 $R(\cdot, \cdot)$ (recall), 302
 T (ontology types), 35
 V (ontology values), 35
 $W(\cdot, \cdot)$ (weighted harmonic mean), 315
 $\Delta(\cdot, \cdot)$ (linkage measures), 119
 Λ (set of alignments), 46
 Ω (set of ontologies), 46
 Σ (synonym resource), 100
 Θ (correspondence relations), 43
 Ξ (confidence structure), 44
 α (alignment), 48
 α -closure, 52, 309
 α -consequence, 52, 309
 \perp (ontology exclusion), 35
 δ (dissimilarity or distance), 86
 ϵ

- empty string, 87

iteration threshold, 135
 \in (ontology instantiation), 35
 κ (confidence function), 44
 \models

- correspondence satisfaction, 51
- ontology entailment, 37
- satisfiability of a formula, 36

 $\omega(\cdot, \cdot)$ (alignment proximity), 307
 π (probability), 102
 σ (similarity), 85
 \sqsubseteq (ontology specialisation), 35

A

Accuracy (matching -), 304
Agent, 3, 15, 19, 23, 391

- cognitive -, 19
- communication, 24, 382
 - language, 19
 - reactive -, 19

Aggregation

- fuzzy -, 164, 165
- similarity -, 160, 348

AgreementMaker, 229, 230, 274, 280
AGROVOC, 297
ALCOMO, 195, 259, 390
Algebra, 341
Alignment, viii, 39, 42–53, 321

- API, 96, 156, 161, 225, 226, 244, 266, 273, 288, 299, 309, 331, 333, 346, 347, 350, 380, 383, 389, 392

bijective -, 47

coherence, 51, 130, 193, 194, 260, 261, 390

completeness, 302

completion, 293

¹ A searchable index can be found on-line at <http://book.ontologymatching.org>.

- Alignment (*cont.*)
- composition, 384, 394
 - consistency, 51, 66, 130, 192–195, 390, 395
 - correctness, 302
 - debugging, 193–196, 254, 313
 - disambiguation, 193, 313
 - edition, 65, 66, 369–375
 - evaluation, 62, 64–66, 347
 - evolution, 11, 394, 395
 - extraction, 154, 186–192, 348
 - greedy -, 190
 - extractor, 187
 - filter, 187, 192
 - format, 67, 68, 288, 321–337, 346, 349, 390
 - improvement, 192–196
 - inconsistency, *see* alignment consistency
 - initial -, 354
 - injectivity, 47, 120, 187, 190, 294
 - life cycle, 56, 57, 69, 350, 394
 - maximal cardinality -, 190
 - metadata, 61, 65, 67, 329, 337–340, 358, 360, 365, 386, 391, 392
 - identification -, 338
 - provenance -, 338–340
 - qualification -, 340
 - model, 51
 - multiplicity, 47, 293, 294
 - one-to-one -, 47, 75, 190, 191, 207, 235, 236, 238, 239, 245, 251, 264, 294
 - pattern, 130
 - reference -, 300–302, 304, 305, 307
 - relation, 42
 - repair, *see* alignment debugging
 - repository, 61, 66, 67, 390–394
 - retrieval, 60, 61, 67, 350
 - reuse, 56, 60, 67, 79, 81
 - reversible -, 47
 - satisfiability, 51, 145
 - semantics, 48–53
 - server, 61, 67, 347, 350, 392, 393
 - service, 390–394
 - sharing, 67, 68, 390
 - structure, 42–48
 - surjective -, 47
 - total -, 47, 187, 190, 294, 379, 381
 - trimming, *see* threshold
 - update, 293
 - validity, 51
- Alterator, 299
- AlViz, 373, 374
- Ambient computing, 19, 20, 391
- AMC, 268, 282
- AMS, 182, 268, 269, 282
- Anchor, 39, 121, 127, 145, 148, 150–152, 157, 159, 187, 209, 223, 224, 229, 230, 248, 261, 274, 313, 353
 - reduction, 152
- Anchor-Flood, 152, 228, 229, 274, 280
- Anchor-Prompt, 127, 208, 209
- Anchoring, 157, 224
- Annotation, *see* metadata (alignment -)
- Antipattern, 130, 193, 195, 254
- Antonym, 99
- AOAS, 223, 273, 279
- APFEL, 262, 263, 282, 348, 356
- Application
- specific evaluation, 290, 308, 314
 - ontology, 32
- Approximate algorithm, 74
- Approximation (ontology -), 48
- Argumentation, 15, 362, 364, 365
- AROMA, 62, 65, 152, 252, 253, 267, 276, 281
- Array, 112
- Artemis, 74, 206, 207, 271, 278
- ArtGen, 206
- Articulation axiom, *see* bridge axiom
- ASCO, 216, 217, 272, 279
- ASMOV, 195, 254, 255, 276, 281, 303
- Association rule, 252
- Associativity, 161
- ATOM, 343
- Automatch, 236, 237, 274, 280
- Automation (evaluation -), 290, 291
- AutoMed, 201
- Autonomy
- design -, 13
 - participation -, 13
 - total -, 13, 14
- Average, 166, 264
- linkage, 119
 - ordered weighted -, 166, 260, 263, 266, 269
 - weighted -, 164, 165, 184, 244, 245, 315
- Axiom (bridge -), 5, 40, 379, 380, 390
- B**
- Background knowledge, 156, 157
- Bag, 112
- of words, 93, 97, 217, 248
- Bagging, 184
- Bandwidth, 312
- Bayesian
- classification, 364, 366
 - learning, 173, 174, 182, 183, 195, 235–238, 241, 242, 244, 245, 249, 251, 255, 257, 271, 384
 - network, 140–143, 219, 220

- BayesOWL, 219, 220, 273, 279
Beam search, 236
BeMatch, 227, 273, 279
Benchmark, 286
 competence -, 289
 suite, 286
Best match, 190–192
BibSter, 14
Bijective alignment, 47
BioPortal, 4, 345, 391–393
BizTalk schema mapper, 377
Blank normalisation, 88
Blocking, 117, 150, 386
BLOOMS, 99, 225, 226, 273, 279
 BLOOMS+, 225, 226, 279
BN mapping, 219, 220
Boosting, 184, 267
Boundary condition, 161
Bounded path matcher, 126
Boundedness (evaluation measure -), 308
Bridge
 axiom, 5, 24, 40, 379, 380, 389, 390
 concept - (in MAFRA), 323
 property - (in MAFRA), 323
 rule, 323, 326, 380
 semantic - (in MAFRA), 323
Brown corpus, 103
Browsing (semantic web -), 20–23
Built-in composition, 354
- C**
C-OWL, 326, 327, 336, 337, 383, 390, 396
CAIMAN, 234, 274, 280
Cardinality, 80
 compatibility, 110, 212
 maximal -, 190
 property -, 107, 109–111
Case normalisation, 88, 217
Catalogue integration, viii, 3, 6, 8, 9
CATCH, 393, 394
CBW, 221, 222, 273, 279
Chebichev distance, 162
Chimaera, 372, 373
CIDER, 224, 226, 267, 273, 279, 347
City-blocks distance, *see* Manhattan distance
Class, 34
 exclusion, 35
 specialisation, 35
Classification, 27, 28, 214, 234, 237, 245,
 278–280
 of matching approaches, 75–83, 201, 203,
 271
Clio, 383, 384
Closure (α -), 52, 309
CODI, 260, 277, 282
CogZ, 374, 375
Coherence (alignment -), 51, 130, 193, 194,
 260, 261, 390
Collective matching, 357–360
COMA, 74, 210, 211, 244, 272, 278, 342–344
 COMA++, 210, 211, 256, 272, 278,
 342–344, 354
Combination (matcher -), *see* composition
Common homomorphic directed subgraph,
 122
Communication (agent -), 19, 382
Commutativity, 161
Comparison evaluation, 290
Compatibility
 cardinality -, 110
 data type -, 108, 110, 207, 218, 227, 246
 transformation -, 388
Competence benchmark, 289
Complete linkage, 119
Completeness (alignment -), 302
Completion (alignment -), 293
Composition, 76
 built-in -, 354
 manual matcher -, 354, 355
 opportunistic -, 354
 parallel -, 154, 245
 heterogeneous -, 155
 homogeneous -, 155
 sequential -, 153, 205, 206, 209, 217, 238,
 245
 user-driven -, 354
 web service -, 16, 17, 40, 340
Compound similarity, 160
Computing
 ambient -, 19, 20
 pervasive -, *see* ambient computing
Concatenation of strings, 87
Concept
 bridge (in MAFRA), 323
 lattice, 114
Conceptual model, 31, 32
Conceptualisation mismatch, 38
Conditional probability table, 141, 220
Confidence, 75
 degree, 44
 structure, 44
Consequence
 α -, 52, 309
 ontology -, 37
Consistency
 alignment -, 51, 66, 130, 192–195, 390, 395
 network of ontologies -, 51
Constraint-based technique, 76, 79, 80, 107

- Content-based matching, 74, 78
 ContentMap, 147, 195, 266, 267, 282
 Context, 74
 -based matching, 74, 78, 80, 81, 156–160,
 212, 224, 270
 Contextualising, 157
 Continuity
 evaluation -, 286
 property, 164
 Convergence (fixed point algorithm -), 132,
 135, 137
 Corpus, 103
 -based similarity, 103
 Brown -, 103
 Correctness (alignment -), 302
 Correspondence, 39, 43
 analysis, 94
 graded -, 75
 justified -, 338
 pattern, 129, 130
 satisfiability, 50
 Cosine similarity, 93, 96
 Cosynonymy similarity, 102
 Count, 328
 Cover (graph -), 190
 Coverage, 38, 49
 Crosslingual matching, 105
 CSR, 180, 257, 258, 276, 282
 CtxMatch, 213, 214, 271, 272, 278
 Cupid, 74, 209, 210, 211, 251, 272, 278
 Cyc, 32, 224
- D**
- Damerau–Levenshtein distance, 91
 DAML+OIL, 323
 Data
 analysis technique, 80, 82
 integration, 3, 5, 9–13, 15, 24, 49, 153,
 239, 336, 341, 348, 382, 389
 interlinking, 12, 23, 24, 40, 59, 150, 288,
 333, 384–387
 linked -, 11–13, 23, 40, 59, 225
 set (evaluation -), 286, 291–299
 transformation, 381
 translation, viii, 24, 40, 58, 324, 333, 341,
 381–384, 387, 390
 translator, *see* data translation
 type, 29, 34, 74, 107, 111, 234
 compatibility, 108, 110, 207, 218, 227,
 246
 value, 34
 warehouse, 3, 5
 web of -, *see* linked data
- Database
 federated -, 8
 schema, 25, 28, 29, 108, 203, 213,
 236–238, 242, 251, 269, 341, 383,
 395
 matching, viii
 Datalog, 240, 241
 DCM, 220, 221, 273, 279
 DDL, 390
 Debugging (alignment -), 193–196, 254, 313,
 357
 Decision tree, 174, 178, 179, 182, 183, 228,
 242, 245, 257, 259, 275, 279
 learning, 178, 263
 Deep web matching, 23, 247
 Definiteness (property), 86
 Degree
 confidence -, 44
 of completeness, 302
 of correctness, 302
 DELTA, 203, 271, 278
 Delta threshold, 188
 Dempster combination rule, 168
 Dependency
 graph, 238, 362, 366, 366, 367
 tuple-generating -, 44, 384
 Descendant similarity inheritance, 230
 Description logic, 145, 328
 technique, 82, 146, 147, 213
 Design autonomy, 13
 Diacritic suppression, 88
 Diagnosis, 194, 196, 357
 globally optimal -, 194, 195
 locally optimal -, 194, 195
 Dice coefficient, 93, 96
 Dictionary, 97, 99
 Digit suppression, 88
 DIKE, 204, 205, 271, 278, 380
 Directory, 27, 28
 Disambiguation
 alignment -, 193, 313
 page, 99
 word sense -, 99, 105, 106, 224
 Disjointness, 35, 75
 Dissemination (evaluation -), 286
 Dissimilarity, 86, 191
 Leacock–Chodorow -, 106, 125
 Distance, 86
 aggregation, *see* similarity aggregation
 Chebichev -, 162
 City-block -, *see* Manhattan
 Damerau–Levenshtein -, 91
 edit -, 90–92, 209, 211, 214, 233, 246, 251,
 264, 303, 304

- Distance (*cont.*)
- edit - (on trees), 127, 215
 - Euclidean -, 93, 110, 162, 163, 238
 - Hamming -
 - on alignments, 301
 - on multisets, 93
 - on sets, 93, 113, 163
 - on strings, 89
 - Hausdorff -, 119
 - Levenshtein -, 91, 161
 - Manhattan -, 93, 162, 163, 164
 - Minkowski -, 93, 162
 - multidimensional -, 162–164
 - n*-gram -, 210, 214, 251, 264
 - Needleman–Wunch -, 91
 - on sequences, 95
 - path -, 95
 - relative-size -, 110
 - tree -, 124
- Distributed
- database, 8
 - knowledge, 53
 - system, 45
- Divergence (Kullback–Leiber -), 94
- Document frequency (inverse -), 94
- DOLCE, 32, 156, 224
- Domain
- specific ontology (technique based on -), 81
 - of interpretation, 36, 50
 - ontology, 32, 40
 - property -, 107, 108, 128
- DPLL procedure, 339, 367–369
- Drago, 383, 390
- DSSim, 22, 231, 274, 280
- DTD, 29, 111, 212, 331
- Dublin core, 83
- Dumas, 238, 239, 274, 280
- DWQ, 201
- E**
- E-commerce, 8–13
- ECOMatch, 182
- Edge count similarity, *see* structural topological dissimilarity
- Edit distance, 90–92, 209, 211, 214, 233, 246, 251, 264, 303, 304
- on trees, 127, 215
- Edition
- alignment -, 65, 66, 369–375
 - ontology -, 4, 5
- EDOAL, 129, 331–333, 336, 337, 346, 378, 382, 389
- Effort-based
- precision, 308
 - recall, 308
- Element-based technique, 76, 78, 107
- Elementary matchers, 76
- Emergent semantics, 15, 16
- Empty
- phrase, 104
 - word, 104
- Engineering (ontology -), 3–5, 55, 68
- Entailment, 48
- Enterprise information integration, *see* data integration
- Entity
- relationship model, 31, 32, 74, 203, 205
 - interpretation, 42
 - language, 42, 46
 - ontology -, 34–36
- EON, 288, 294
- Equalising
- function, 50
 - semantics, 50, 390
- Error minimisation, 356
- ETuner, 181, 264, 265, 282
- Euclidean distance, 93, 96, 110, 162, 163, 238
- Evaluation, 62, 64–66, 285–317
- application-specific -, 290, 308, 314
 - automation, 290, 291
 - comparison, 290
 - type, 289, 290
- Evolution
- alignment -, 11, 394, 395
 - ontology -, 5, 56, 59, 348
- Exact algorithm, 74, 78
- Exclusion, 35
- Exclusivity, 190
- Executability, 336
- Exhaustivity, 75
- Expectation maximisation, 137–139, 256
- Explicitation mismatch, 38
- Expressiveness, 335
- Extendibility, 336
- Extensional technique, 78, 112–120, 234
- External
- resource, 293
 - structure-based technique, 106, 121–130
 - technique, 74
- Extract–Transform–Load (ETL), 384
- Extraction (alignment -), 186–192, 348
- Extractor (alignment -), 187
- Intrinsic linguistic technique, 80, 98–104

- F**
- F-logic, 292
 - F-measure, 181, 265, 304, 306, 315
 - optimal threshold, 305
 - FaCT, 213
 - Falcon-AO, 151, 247–249, 266, 275, 281, 303, 331, 354
 - Fallout, 303
 - False
 - negative, 302
 - positive, 99, 302
 - FCA, *see* formal concepts analysis
 - FCA-merge, 234, 235, 274, 280, 380
 - Feature path equations, 328
 - Federated database, 8
 - Feedback (relevance -), 355–357
 - Filter
 - alignment -, 187, 192
 - similarity -, 187
 - FIPA ACL, 19, 393
 - Fixed point, 53, 132
 - computation, 130–137, 212, 247, 271
 - FMA, 32, 156, 157, 296, 392
 - FOAF, 4
 - FOAM, 156, 243, 244, 256, 262, 263, 266, 276, 331, 347–349
 - Folksonomy, 25–27, 334
 - Formal
 - concept analysis, 114, 234, 235, 243
 - resource-based technique, 79–81
 - Foundational ontology, 32, 130
 - FSM, 241
 - Fuzzy aggregation, 164, 165
- G**
- Galen, 296
 - Galois
 - connection, 114
 - lattice, *see* concept lattice
 - Gap threshold, 188
 - GATE, 97
 - GAV, *see* global-as-view
 - GBM, 241, 242
 - GEM, 256, 257
 - Optima, 256, 257, 276
 - Optima+, 282
 - Generation (test -), 298, 299
 - Genetic programming, 117, 184–186, 226, 232
 - GeRoMeSuite, 159, 222, 223, 273, 279, 342
 - GLAV, *see* global-local-as-view
 - Global
 - as-view, 10, 206
 - local-as-view, 10, 11, 240
 - knowledge, 52
- maximal - similarity, 190
 - Globally optimal diagnosis, 194, 195
 - Gloss, 100, 103, 104, 215, 334
 - overlap, 103, 106
 - GLUE, 74, 180, 235, 236, 274, 280
 - GOALS, 266, 282
 - Gold standard, *see* reference alignment
 - GOMMA, 343, 344
 - Gotoh distance, 92, 96
 - Graded correspondence, 75
 - Granularity, 38, 49
 - matcher -, 76
 - Graph
 - based technique, 79
 - cover, 190
 - dependency -, 366, 367
 - matching, 132, 190
 - maximum weight -, 191, 239
 - minimum weight -, 191, 238
 - Greedy alignment extraction, 190
 - Ground truth, *see* gold standard
- H**
- H-Match, 206–208, 271, 278, 354
 - Hamming distance
 - on alignments, 301
 - on multisets, 93
 - on sets, 93, 113, 163
 - on strings, 89
 - HAMSTER, 255, 276, 281
 - Hard threshold, 188
 - Harmonic
 - adaptive weighted sum, 165, 166
 - mean, 304, 315
 - weighted -, 315
 - Harmony, 349, 349
 - Hausdorff distance, 119
 - HCONe, 215, 216, 272, 278, 331
 - Hermit, 390
 - Heterogeneity, vii, 3
 - conceptual -, 38
 - language -, 292
 - pragmatic -, 38
 - semantic -, 38
 - semiotic -, 38
 - syntactic -, 37
 - terminological -, 38
 - Heterogeneous parallel composition, 155
 - Hitting set, 194
 - Homogeneity, 75
 - Homogeneous parallel composition, 155
 - Homolonto, 230, 230, 231, 274, 280
 - Homonym, 87, 99, 205
 - Horn clause, 44, 327, 328

- HSM, 221, 273, 279
HTTP, 393
Hungarian method, 191
Hypernym, 97, 99, 100, 102, 215
Hyponym, 80, 97, 99, 102, 205, 215
- I**
I3CON, 288
IceQ, 246, 247, 275, 281
IDDL, 390
Ideal semantic
 precision, 309
 recall, 309
Idempotency, 164
Identification metadata, 338
IF-Map, 243, 275, 281
IIMB, 299
ILIADS, 253, 276, 281
Illinois Semantic Integration Archive, 297
IMAP, 236, 274, 280, 361, 362, 364, 366, 367
IMapper, 251, 252, 276, 281
IMatch, 233, 274, 280
IMerge, 371, 372
Implicit matching, 357
Import (ontology -), 4, 5
Improvement (alignment -), 192–196
Inconsistency
 alignment -, *see* alignment consistency
 network of ontologies -, *see* network of
 ontologies consistency
Individual, 34
Inference Web, 361
InfoMix, 201
Informal resource-based technique, 79, 80
Information
 -theoretic similarity, 103, 106
 integration, 3, 5–13, 22
 retrieval, 244
InfoSleuth, 201
Initial alignment, 354
Injectivity (alignment -), 47, 120, 187, 190,
 294
Input, 76
 dimensions, 74
 kind of -, 78
Instance, *see* individual
 -based technique, 74, 76, 79, 82
 matching, 288
Instantiation, 35
Integer linear programming, 260
Integration
 catalogue -, 3, 6, 8, 9
 data -, 3, 5, 9–13, 15, 24, 336, 341, 348,
 382, 389
 information -, 3, 5–13, 22
 ontology -, 3, 40, 340
 schema -, 3, 5, 8, 24
Integrity constraint, 112
Intelligibility (evaluation -), 286
Interlinking (data -), 12, 23, 24, 40, 59, 150,
 288, 333, 384–387
Internal
 structure-based technique, 106–112, 117,
 208
 technique, 74
Interoperability, *see* heterogeneity
Interpretation
 domain of -, 36
 entity -, 42
 ontology -, 36, 42
Intrinsic linguistic technique, 80, 97, 98
Inverse document frequency, 94
IPrompt, 373
Iterative propositional fitting, 220
- J**
Jaccard similarity, 96, 103, 111, 113, 125, 220,
 231, 253, 255, 260
Jaro measure, 92, 96
Jaro–Winkler measure, 92, 96, 217, 223, 228,
 253, 259, 260, 266, 386
JDBC, 393
Jeffrey rule, 220
Jiang–Conrad similarity, 103
Justification, 338
Justified correspondence, 338
- K**
k-nearest neighbours, *see* nearest neighbours
KAON2, 348
Key, 107, 108, 115
KIF, 243, 292
Kind of input, 78
Knob, 265
KnoFuss, 117, 385, 386
Knowledge base, 33
Knowledge web, xii
Kraft, 201
Kullback–Leiber divergence, 94
- L**
Language
 -based technique, 79, 80, 96–106, 247
 independence, 335
 ontology -, 33–37
 query -, 42
Latent semantic indexing, 94, 216
LAV, *see* local-as-view
LCS, 165, 263, 282

- Leacock–Chodorow dissimilarity, 106, 125
 Learning, 171–180
 Lemmatisation, 97, 245
 Levenshtein distance, 91, 96, 161, 217, 222, 223, 226, 228, 229, 254, 258–260
 Lexicon, 99, 106
 - interlingual –, 105
 - semantico-syntactic –, 99
 Life cycle (alignment –), 56, 57, 69, 350, 394
 Lily, 152, 229, 274, 280, 303
 Linguistic
 - quantifiers, 165, 260, 263
 - technique, 76, 107, 206–210, 217, 245, 248
 - based on - resource, 205–207, 209–213, 215, 272
 Link stripping, 88
 Linkage
 - dissimilarity
 - average –, 119
 - complete –, 119
 - single –, 119, 217
 - record –, 117
 Linked data, 11–13, 23, 40, 59, 225
 Linkkeys, 115
 LinQuer, 387
 List, 112
 Local
 - as-view, 10, 11, 15
 - knowledge, 52
 Locally optimal diagnosis, 194, 195
 Logical mismatch, 38
 LogMap, 62, 65, 152, 195, 260–262, 271, 390
 - LogMap2, 277, 282
 LOV, 59, 61
 LSD, 83, 235, 236, 251, 274, 280
 Lucene, 97
- M**
- MAFRA, 323, 324, 336, 337, 344, 345
 Magpie, 21
 Majority
 - vote, 166
 - weighted vote, 167
 Manhattan distance, 93, 96, 162, 163, 164
 MapForce, 377
 MapOnto, 213, 272, 278, 383
 Mapping, 39, 179
 - in Drago, 390
 - in model management, 341
 - rule, 39, 179
 MapPSO, 231, 232, 274, 280
 MapReduce, 151, 387
 MapSSS, 303
- Markov
 - logic network, 143, 260
 - network, 143, 233, 270
 Marriage (stable –), 190–192
 Match-based similarity, 119
 Matcher
 - composition, 153–156
 - manual –, 354, 355
 - selection, 61–63
 Matching, 39
 - accuracy, 304
 - coefficient, 93, 96
 - collective –, 357–360
 - crosslingual –, 105
 - graph –, 132
 - implicit –, 357
 - instance –, 288
 - memory consumption, 312
 - monolingual –, 105
 - multilingual –, 105
 - multiple –, 41, 46
 - process, 41, 42
 - scalability, 312
 - schema –, 55, 107
 - speed, 312, 317
 - usability, 313
 MatchPlanner, 182, 228, 273, 279
 Matrix, 86, 154
 Maximality
 - evaluation measure –, 308
 - of similarity measures, 85
 Maximum
 - weight matching, 191, 239
 Mean (harmonic –), 304
 Measure
 - Jaro –, 92
 - Jaro–Winkler –, 92, 386
 - SMOA –, 93, 96, 223, 224, 232, 248, 261
 Mediation, 24, 40
 - query –, 57, 59, 68, 324, 387, 387, 388–390
 Mediator, 5, 40, 336, *see* mediation
 Memory consumption, 312
 Mereologic structure, 124, 127
 Merging (ontology –), viii, 4, 24, 39, 40, 57–59, 68, 208, 209, 215, 216, 235, 246, 322, 324, 328, 345, 348, 372, 378–381
 Meronym, 100, 207, 246
 Message translation, 24
 Metadata (alignment –), 61, 65, 67, 169, 329, 337–340, 358, 360, 365, 386, 391, 392, 394
 Method
 - composition, 153–156

- learning, 171–180
Methodology, 69
Metric, 86
Minimal
 inconsistency preserving subalignment, 194, 196
 semantic matching, 215
 unsatisfiability preserving subalignment, 194, 196
Minimality, 86
Minimisation (error -), 356
Minimum
 cost maximum flow, 237
 weight matching, 191, 238
Minkowski distance, 93, 162
MIPS, *see* minimal inconsistency preserving subalignment
Mismatch, *see* heterogeneity
 conceptualisation -, 38
 explication -, 38
 logical -, 38
Miss, 304
Mixed technique, 74
MoA, 216, 272, 279
Moda, 342
Modal logic satisfiability, 146
Model
 -based technique, *see* semantic technique
 -theoretic semantics, 78
 alignment -, 51
 conceptual -, 31, 32
 entity–relationship -, 31, 32, 203, 205
 in model management, 341
 in Rondo, 342
 management, 341–344
 of a network of ontologies, 51
 ontology -, 37
ModelGen, 342
Modularisation (ontology -), 151
Monge–Elkan distance, 92, 96, 217, 231
Monolingual matching, 105
Monotony, 161
 increasing, 164
Morphism, 48
 in Rondo, 342
Morphological
 analysis, 98
 normalisation, 210, 293
Mostro, 382
MoTo, 165, 259, 260, 277, 282
MSeer, 265, 266, 282, 361
Multi
 -response linear regression, 183
 alignment, 46
dimensional distance, 162–164
lingual, 97
 matching, 105
 technique, 104, 105
plicity
 property -, 107, 109–111
 similarity, 111
 set, 93, 112
MultiFarm, 297
Multiple matching, 46, 293
Multiplicity, 80
 alignment -, 47, 293, 294
MUPS, *see* minimal unsatisfiability preserving subalignments
Muse, 371
MWSDI, 218, 273, 279
- N**
- n*-gram, 96
 distance, 210, 214, 251, 264
 similarity, 90, 177
NAL, 297
Name-based technique, 78, 87–107, 244, 366
Natural language processing, 80
NCI, 296
Nearest neighbours, 174, 241, 249, 275
Needleman–Wunch distance, 91, 96
Negative
 false -, 302
Neighbours (nearest -), 174, 241, 249, 275
NeOn Toolkit, 349, 350, 391
Network of ontologies, 40, 46, 68, 346, 404
 consistency, 51
 inconsistency, *see* network of ontologies
 consistency
 model of -, 51
Neural network, 175–177, 183, 243, 263
NIST, 287, 288
Noise, 303
NOM, 243, 244, 275, 281, 348
Norm (triangular -), 132, 160–162, 165
Normalisation, 244, 245
 measure, 86
 morphological -, 210, 293
 string -, 88, 89
 blank -, 88
 case -, 88
 diacritic -, 88
 digit -, 88
 link stripping -, 88
 punctuation -, 89

- O**
- OAEI, 60, 62, 287–289, 291, 296, 297, 303, 317, 331, 347
 - Object, *see* individual
 - oriented model, 74
 - categorical -, 48
 - identifier, 342
 - Observer, 201
 - Occurrence
 - of a character in a string, 87
 - of a substring in a string, 87
 - ODEDialect, 322
 - OID, 342
 - OLA, 134, 247, 267, 271, 275, 281, 331, 347
 - algorithm, 134–137
 - OMMap, 244, 244, 245, 275, 281, 331, 347
 - OMEN, 220, 220, 273, 279
 - OMviaUO, 157, 224, 224, 225, 273, 279
 - OMWG, 331
 - One-to-one alignment, 47, 75, 190, 191, 207, 235, 236, 238, 239, 245, 251, 264, 294
 - ONION, 205, 205, 206
 - Onto function, 47
 - ONTOBI, 299
 - OntoBuilder, 209, 209, 272, 278, 380
 - OntoFarm, 297
 - Ontologies (network of -), 40, 46, 68, 404
 - Ontology, 25, 32, 33
 - approximation, 48
 - consequence, 37
 - domain -, 40
 - edition, 4, 5, 380
 - engineering, 3–5, 24, 55, 68
 - entity, 34–36
 - language, 42, 46
 - evolution, 5, 56, 59, 348
 - foundational -, 130
 - import, 4, 5
 - integration, 3, 40, 340
 - interpretation, 36, 42
 - language, 33–37, 292
 - management, 56
 - merging, viii, 4, 24, 39, 40, 57–59, 68, 208, 209, 215, 216, 235, 246, 322, 324, 328, 345, 348, 372, 378, 378, 379–381
 - model, 37
 - modularisation, 151
 - partition, 39, 149–152, 231, 248, 257, 261, 268
 - peer-to-peer -, 15
 - reconciliation, 40
 - satisfaction of a formula, 36
 - semantics, 36, 37
 - syntax, 35
 - transformation, 24, 40, 328, 336, 380, 380, 381, 387
 - translation, 40, 55, 345, 380
 - upper-level -, 55, 381
 - version, 5, 40
- Ontology Alignment Evaluation Initiative, *see* OEAII
- Ontologymatching.org, 297, 377
- OntoMas, 62
- OntoMediate, 358
- OntoMerge, 380, 380
- OntoSim, 232
- Ontowrap, 347
- Opportunistic composition, 354
- Orchid, 384
- Ordered weighted average, 166, 166, 260, 263, 266, 269
- Oriented
 - precision, 308
 - recall, 308
- Origin, 76
 - dimension, 74
- Output dimension, 75
- Overall, 304, 305, 308
- Overlap, 96
- OWA, *see* ordered weighted average
- OWL, 15, 33–37, 43, 60, 68, 74, 104, 106, 108, 109, 111, 112, 123, 134, 147, 207, 209, 213, 216, 231, 233, 244, 247, 256, 263, 267, 269, 278, 279, 292, 296, 321–328, 331, 333, 336, 337, 347, 373, 378, 380, 389, 390, 396
 - DLP, 348
 - C-OWL, 326, 327, 390
 - sameAs, *see* sameAs
 - SWRL, 327–329, 390
- P**
- P2P, *see* peer-to-peer
- Parallel composition, 245
- Parameter, 262, 264, 293, 354
- PARIS, 262, 262, 277, 282
- Part-of-speech tagger, 97, 106
- Partially ordered synonym resource, 100, 100, 103
- Participation autonomy, 13
- Particle swarm optimisation, 139, 140, 231
- Partition (ontology -), 39, 149–152, 231, 248, 257, 261, 268
- Path, 324
 - bounded - matcher, 126
 - distance, 95

- Pattern
-based matching, 81, 128–130
alignment -, 130
anti-, 130, 193, 195, 254
correspondence -, 129, 130
redundancy -, 215
refinement -, 130, 232
Peer (in Drago), 390
Peer-to-peer, 13–16, 24, 207, 389
ontology, 15
system, 3
semantic -, 14
Pellet, 147, 213, 390
Percentage threshold, 188
PerMIS, 288
Perspective, 38, 49
Pervasive computing, *see* ambient computing
Picsel, 201
Pivot language, 105
Polysemy, 87
Pooling, 311, 316
PORSCHE, 152, 227, 228, 228, 273, 279
Positive
false -, 99, 302
true -, 99, 302
Positiveness, 85
evaluation measure -, 308
Pre-similarity, 86, 90
Precision, 65, 301, 302, 305–310, 315
/recall curve, 303, 347
@ k , 303
effort-based -, 308
generalised -, 305–310
ideal semantic -, 309
oriented -, 308
relative -, 310, 311
relaxed -, 307
semantic -, 309
symmetric -, 308
weighted -, 306
Prior+, 258, 276, 282
ProbaMap, 242
Process
dimension, 74
trace, 339
Product
classification, 9
weighted -, 161, 315
Prolog, 243
Prompt, 5, 345, 346, 373
Anchor-Prompt, 208, 209, 272, 345
iPrompt, 345, 373
PromptDiff, 209, 345, 346
PromptFactor, 345
Proof Markup Language, 361
Property, 107
bridge (in MAFRA), 323
Proportional threshold, 188
Propositional
satisfiability, 82, 145, 146
technique, 145, 146
Protégé, 5, 345
Provenance metadata, 338–340
Prüfer sequences, 219
Pruning (search-space -), 152, 228, 229, 258
Punctuation
normalisation, 217
suppression, 89
Purpose independence, 336
- Q**
- QOM, 74, 243, 244, 263, 275, 281, 348
Qualification metadata, 340
Quality (evaluation -), 286
Query
answering, viii, 23, 24, 40, 348
language, 42
mediation, 57, 59, 68, 324, 387, 387,
388–390
transformation, 24, 59, 68, 322, 383
QuickMig, 211, 212, 272, 278
- R**
- Range (property -), 107, 108, 128
RDF, 6, 14, 68, 74, 243, 248, 256, 263, 333,
335, 347, 382, 385, 387, 389
/XML, 331, 335, 347
schema, 15, 209, 217, 279, 292, 334, 335,
344, 389
RDFS, *see* RDF Schema
Reasoning, 39, 380, 389, 390
Recall, 65, 301, 302, 305–310, 315
effort-based -, 308
generalised -, 305–310
ideal semantic -, 309
oriented -, 308
relative -, 310, 311
relaxed -, 307
residual -, 311
semantic -, 309
symmetric -, 308
weighted -, 306
Recall+, 311
Reconciliation (ontology -), 40
Record linkage, 117
Reduced semantics, 50, 378, 390
Redundancy pattern, 215
Reference alignment, 300–302, 304, 305, 307

- Refinement pattern, 130, 232
- Relation, 34
alignment -, 42
- Relational
database model, 74
structure-based technique, 106, 121–130, 205–207, 210, 212, 217, 245, 247, 248, 272
- Relative
precision, 311
recall, 311
size distance, 110
- Relaxed
precision, 307
recall, 307
- Relevance feedback, 263, 355–357
- Repair
alignment -, *see* alignment debugging plan, 147, 267
- Repository
alignment -, 61, 66, 67, 390–394
of structure, 81
- Residual recall, 311
- Resnik similarity, 103, 106
- Resource, 293
consumption, 312
- REST, 393
- Retrieval (alignment -), 60, 61, 67, 350
- Reuse
-oriented matcher, 83, 210–212, 272
alignment -, 56, 60, 67, 79, 81
- Reversible alignment, 47
- RIF, 328, 328, 329
- RiMOM, 249, 250, 275, 281, 303, 354
- ROC curves, 347
- ROCK, 151
- Role-value map, 328
- Rondo, 132, 156, 188, 212, 341–343, 354, 380, 383
- Rule
bridge -, 323, 326, 380
mapping -, 39, 179
- Rule interchange format, *see* RIF
- RuleML, 327
- S**
- S-Match, 74, 213–216, 271, 272, 361–369
- SAMBO, 252, 252, 276, 281
- SameAs (owl : sameAs), 11, 12, 68, 385, 387
- Sampling, 310, 311
- SAT, *see* propositional satisfiability
modal, *see* modal logic satisfiability
solver, 145, 146, 214, 362, 367
- SAT4J, 215, 367–369
- Satisfaction (by an ontology), 36
- Satisfiability
alignment -, 51, 145
modal logic -, 145, 146
of correspondence, 50
propositional logic -, 145, 146, 369
- Saturation, 75
- SBI&NB, 237, 238, 274, 280
- Scalability, 312
- Scarlet, 157, 159, 223, 224, 273, 279
- Schema
-based technique, 74, 76
database -, 25, 28, 29, 203, 213, 236–238, 242, 251, 269, 341, 383, 395
integration, 3, 5, 8, 24, 39
matching, 55, 107
RDF -, 15, 209, 217, 279, 334, 335, 344
UML, 25
XML -, 15, 29–31, 109, 213, 214, 235, 245, 278, 280, 292, 328, 341, 383, 395
- Schemr, 5, 349
- Scope, *see* perspective
- SEALS, 291
- Search
-space pruning, 152, 228, 229, 258
beam -, 236
- SecondString, 96
- SEKT mapping language, 331, 336
- Selection (matcher -), 61–63
- Selector (in Rondo), 342
- Semantic
bridge (in MAFRA), 323
ideal - precision, 309
ideal - recall, 309
peer-to-peer system, 14
precision, 309
recall, 309
technique, 74, 78, 79, 81, 145–148, 213, 361, 380
web
browsing, 20–23
service, 16, 17, 382
- Semantic Bridge Ontology, 323, 324, 336
- Semantic Web Rule Language, *see* SWRL
- SemanticIntegrator, 377
- Semantics
alignment -, 48–53
emergent -, 15, 16
ontology -, 36, 37
- SeMap, 253, 254, 276, 281
- SEMINT, 242, 243, 275, 281
- SeqDisc, 218, 219, 273, 279

- Sequential composition, 205, 206, 209, 217, 238, 245
Service
(in MAFRA), 323
web -, 3
composition, 40
semantic -, 16, 17, 24, 56, 382
Set, 112
SGen, 299
Sharing (alignment -), 67, 68, 390
Sibling similarity contribution, 230
Sigmoid, 189, 244, 257
Silence, 303
Silk, 117, 347, 377, 385–387, 396
Similarity, 85, 191
aggregation, 160–165, 348
compound -, 160
corpus-based -, 103
cosine -, 93
cosynonymy -, 102
filter, 187
global - computation, 130–137
gloss-overlap -, 103
information-theoretic -, 103, 106
Jaccard -, 103, 111, 113, 220, 231, 253, 255, 260
match-based -, 119
multiplicity -, 111
n-gram -, 90, 177
nonsymmetric -, 86
pre-, 86, 90
Resnik -, 103, 106
structural topological -, 102, 106, 124
substring -, 90
synonymy -, 101
upward coticopic -, 126
vector-based -, 93, 240
Wu–Palmer -, 102, 106, 125, 151, 259
Similarity flooding, 132–134, 137, 212, 223, 227, 229, 259, 272, 278, 342
SimPack, 96, 106
Simplicity, 335
SIMS, 201
Single linkage, 119, 217
Singular value decomposition, 94
SKAT, 205, 206, 271, 278, 380
SKOS, 68, 104, 297, 321, 322, 325, 333–336, 394
mapping, 394
SMART, *see* Prompt
Smart Matcher, 255, 256, 276, 282
SMB, 267, 268, 282
Smith–Waterman measure, 92, 96, 257, 259
SMOA measure, 93, 96, 223, 224, 232, 248, 261
SNOMED, 296
SOAP, 393
Solution space, 399
SomeWhere, 15
Soundex, 96
SPARQL, 68, 333, 347, 382, 383, 386, 389, 396, 430
SPEC, 290
Specialisation, 35, 81
Speed, 312, 317
Spicy, 384
SPLMap, 240, 241, 275, 280, 383
SQL, 28, 32, 383, 384, 389, 396
 DDL, 292
Stable marriage, 190–192
Stacked generalisation, 182–184, 235, 251, 259
Statistical technique, 80, 82
STBenchmark, 299
Stemmer, 106
Stemming, *see* lemmatisation
Stopword elimination, 98
Strengthening, 189, 190
String, 87
 -based technique, 79, 88–96, 117, 203–205, 208–210, 214, 217, 239, 241, 244–247, 251, 271, 272, 275, 384
 concatenation, 87
 equality, 89, 95
 normalisation, 88, 89
 occurrence
 of a string, 87
 of a substring, 87
 substring, 87
Stringmetrics, 96
Structural
 technique, 78
 topological dissimilarity, 102, 106, 124
Structure-based
 internal - technique, 107–112, 117
 relational - technique, 106, 121–130, 205–207, 210, 212, 217, 245, 247, 248, 272
 technique, 76, 78, 106–112, 121–130, 244
Stylus, 377
Substring, 87
 similarity, 90
 test, 89
Subsumption, 35, 146, 388
 test, 146

- Sum
 weighted -, 162–165, 210, 212, 217, 236, 315
 harmonic adaptive -, 165, 166
SUMO, 32, 224
Support vector machine, 177, 178, 242, 257, 263
Surjective alignment, 47
Swing, 299
Swoogle, 226
SWRL, 325, 327–329, 336, 337, 347, 380, 390, 396
Symmetric
 precision, 308
 recall, 308
Symmetry, 85
 evaluation measure -, 308
 property -, 107, 112
Synonym, 80, 87, 99, 101, 205, 212, 221
Synonymy similarity, 101
Synset, 64, 99, 100–103
Syntactic technique, 74, 78, 78
Syntax (ontology -), 35
Systematicity (evaluation -), 286
- T**
T-tree, 233, 234, 274, 280
TaxoMap, 151, 232, 232, 233, 274, 280
Taxonomy, 27, 28, 124–127, 236
 -based technique, 79, 81
Term, 97, 99
 extraction, 98
 frequency, 94
 -inverse document frequency, 94, 96, 175, 217, 234, 239, 248
Terminological
 technique, *see* name-based technique
Terminology, 99
Tess, 208, 271, 278
Test generation, 298, 299
TFIDF, *see* term frequency-inverse document frequency
THALIA, 297
Thesaurus, 99, 106, 334, 345
Threshold, 66, 187–190, 205–210, 217, 220, 221, 234, 235, 237, 244, 245, 249, 262, 264, 304, 347, 350, 356
 delta -, 188
 double -, 66
F-measure (optimal -), 305
gap -, 188
hard -, 188, 208
percentage -, 188
proportional -, 188
- Time (processing -), 293
Tokenisation, 97, 216, 344
ToMAS, 383, 394, 395
Top- k , 209
Total
 alignment, 47, 187, 190, 294, 379, 381
 autonomy, 13, 14
Trace (process -), 339
Transformation
 compatibility, 388
 data -, 381
 ontology -, 24, 40, 328, 336, 380, 381, 387
 query -, 24, 59, 68, 322, 383
Transitivity (property -), 107
Translation
 data -, viii, 24, 40, 58, 324, 333, 341, 381–384, 387, 390
 message -, 24
 ontology -, 40, 55, 345, 380
TransScm, 204, 271, 278, 383
TREC, 287, 303
Tree distance, 124
Triangular
 inequality, 86
 norm, 132, 160–162, 165
Trimming (alignment -), *see* threshold
True
 positive, 99, 302
Truth (ground -), *see* gold standard
Tsimmis, 201
Tuning, 180–186, 313
 parameter -, 294
Tuple-generating dependency, 44, 384
- U**
UFO, 32
Ultrametric, 86
 inequality, 86
UML, 25, 31, 110, 292, 293, 324, 337
 model, 341
UMLS, 28, 214, 223, 252, 254, 260, 273, 276, 277, 296, 392
Update (alignment -), 293
Upper-level ontology, 32, 55, 81, 156, 224, 225, 270, 381
Upward cotopic similarity, 126
URI, 67, 137, 330, 335
Usability, 313
User
 -driven composition, 354
 input, 83, 145, 293, 294, 307, 360
 interaction, 211, 216, 234, 263, 342, 353–375

- V**
Validity (alignment -), 51
Value, 34
Variable, 328
Vector-based similarity, 93, 240
Version, 345
ontology -, 5, 40
Vote
 majority -, 166
 majority weighted -, 167
VSBM, 241, 241, 242
- W**
W3C, 324
Watson, 59
Weakening, 189, 190
Web
 compatibility, 335
 deep -, 23, 247
 of data, *see* linked data
 service, 3, 16, 17
 composition, 40, 340
 semantic -, 17, 24, 56
Weighted
 average, 164, 165, 184, 244, 245, 315
 ordered, 166, 260, 263, 266, 269
 harmonic mean, 315
 precision, 306
 product, 161, 315
 recall, 306
 sum, 162–165, 210, 212, 217, 236, 315
 harmonic adaptive -, 165, 166
Weka, 179, 242, 257, 259, 263
- WHIRL, 182
 learner, 174, 175, 235
Wikipedia, 99, 156, 225
Wise-Integrator, 245, 245, 246, 275, 281, 389
Word, 96
 sense disambiguation, 99, 105, 106, 219, 224
WordNet, 62, 64, 99, 125, 126, 145, 156, 157, 161, 177, 179, 205–207, 212–218, 224, 226–229, 231, 232, 241, 245, 246, 249, 252–254, 257, 259, 260, 266, 270–273, 275, 293, 334, 345, 365
WSML, 337
WSMT, 370, 371
Wu-Palmer similarity, 102, 106, 125, 151, 259
- X**
XClust, 152, 212, 213, 272, 278
XML, 6, 29, 68, 74, 214, 297, 326, 335, 377
 RDF/-, 335
 schema, 15, 25, 29–31, 109, 213, 214, 235, 245, 278, 280, 292, 328, 341, 383, 395
 matching, viii
Xpath, 324
XQuery, 297, 384
XSLT, 68, 347, 383, 384
- Y**
YAM, 258, 259, 276, 282
 YAM++, 180, 258, 259, 276, 282